Z zagadnień ontologicznych informatyki

UNIWERSYTET IM. ADAMA MICKIEWICZA W POZNANIU
SERIA FILOZOFIA I LOGIKA NR 121

IZABELA BONDECKA-KRZYKOWSKA

Z zagadnień ontologicznych informatyki

WYDAWNICTWO NAUKOWE

POZNAŃ 2016

ABSTRACT. Bondecka-Krzykowska Izabela, *Z zagadnień ontologicznych informatyki* [From ontological issues of computer science]. Poznań 2016. Adam Mickiewicz University Press. Seria Filozofia i Logika nr 121. Pp. 266. ISBN 978-83-232-3062-5. ISSN 0083-4246. Text in Polish with a summary in English.

From ontological issues of computer science takes a look at this field through the eye of the philosopher, and contains an analysis of selected ontological issues related to computer science. It also constitutes an attempt to define computer science as a scientific discipline through the prism of the objects it researches. The book presents fundamental problems of philosophy related to the following terms: the computer, computer programmes, information and virtual reality. It analyses their nature, mutual relations and the attempts made to create an ontology for computers, programmes, information and virtual reality as objects of enquiry for computer science. The influence of emerging concepts related to computer science for particular classic concepts and ontological questions is also described.

Izabela Bondecka-Krzykowska, Uniwersytet im. Adama Mickiewicza w Poznaniu, Wydział Matematyki i Informatyki UAM, Zakład Logiki Matematycznej, ul. Umultowska 87, 61-614 Poznań, Poland

Recenzent: ks. dr hab. Adam Olszewski

Publikacja dofinansowana przez
Rektora Uniwersytetu im. Adama Mickiewicza w Poznaniu
oraz Wydział Matematyki i Informatyki UAM

Projekt okładki: Helena Oszmiańska-Napierała

Redaktor: Malwina Błażejczak

Redaktor techniczny: Dorota Borowiak

Łamanie komputerowe: Eugeniusz Strykowski

ISBN 978-83-232-3062-5
ISSN 0083-4246

WYDAWNICTWO NAUKOWE UNIWERSYTETU IM. ADAMA MICKIEWICZA W POZNANIU
61-701 POZNAŃ, UL. ALEKSANDRA FREDRY 10
www.press.amu.edu.pl
Sekretariat: tel. 61 829 46 46, faks 61 829 46 47, e-mail: wydnauk@amu.edu.pl
Dział sprzedaży: tel. 61 829 46 40, e-mail: press@amu.edu.pl

Wydanie I. Ark. wyd. 16,50. Ark. druk. 16,625

DRUK I OPRAWA: EXPOL, WŁOCŁAWEK, UL. BRZESKA 4

Spis treści

Panu Profesorowi
Romanowi Murawskiemu

Wstęp

Informatyka jest stosunkowo młodą dyscypliną naukową, wywodzącą się z matematyki, logiki i inżynierii, zajmującą szczególne miejsce w naszej rzeczywistości. Z osiągnięć informatyki korzystamy na każdym kroku: systemy komputerowe – rozumiane jako połączenie maszyn nazywanych komputerami oraz oprogramowania – stanowią nieodłączną część współczesnego życia. Kolejnym pokoleniom coraz trudniej jest wyobrazić sobie świat bez całej gamy komputerów przenośnych, telefonów komórkowych, Internetu, sklepów on-line, komunikatorów, portali społecznościowych czy też bankowości internetowej. Znacząca część naszego życia związana jest z wytworami informatyki – komputerami oraz z ich oprogramowaniem.

Systemy komputerowe stają się również nieodłączną częścią wielu gałęzi nauki. Wykorzystywane są nie tylko do wykonywania skomplikowanych obliczeń, lecz także do eksperymentowania, a nawet do dowodzenia twierdzeń matematycznych. Mówi się często o nowym paradygmacie w nauce – paradygmacie obliczeniowym lub informacyjnym.

Spoglądając na wszechobecne wytwory informatyki okiem filozofa, zadajemy sobie pytania o naturę samej informatyki, o to, jakie obiekty bada, jakimi metodami posługują się (lub powinni się posługiwać) w tych badaniach informatycy. Pytamy zatem o kwestie filozoficzne związane z informatyką – o to, jakie szczegółowe problemy filozoficzne z nią się wiążą.

Jednym z pierwszych tematów rozważań filozoficznych związanych z informatyką była szeroko rozumiana sztuczna inteligencja. Co to jest sztuczna inteligencja? Jakie są jej rodzaje? Czy istnieje związek, a jeśli tak, to jaki, pomiędzy obliczaniem a poznaniem? Czy komputery mogą myśleć? A jeśli mogą, to jak je rozpoznać? – to tylko niektóre z licznych pytań związanych z tą tematyką. Wydaje się, że również dziś powyższe zagadnienia są często podejmowane przez badaczy, pomimo że istnieje już bogata literatura z nimi związana.

Rozważania filozoficzne dotyczące informatyki nie ograniczają się jednak wyłącznie do badań nad sztuczną inteligencją. Szuka się bowiem również odpowiedzi na pytanie o status informatyki jako nauki, o to, jakie obiekty ona bada, oraz jakimi metodami posługują się informatycy. Bada się również związki pomiędzy podstawowymi obiektami i pojęciami związanymi z informatyką: programem i algorytmem, algorytmem i procedurą, programem i jego specyfikacją, programem i implementacją, modelem i programem, jak również – komputerem i oprogramowaniem. Analizuje się ich definicje i własności. Stawia się pytania o to, czy istnieją dziedziny życia, w których komputery nie powinny podejmować decyzji oraz – czy można decyzjom tym zaufać, kto (lub co) ponosi za nie odpowiedzialność.

Wśród kwestii filozoficznych związanych z informatyką można zatem znaleźć zarówno zagadnienia ontologiczne, jak i metodologiczne czy etyczne. Obecnie powstaje wiele prac filozoficznych dotyczących różnych, często bardzo od siebie odległych, zagadnień. Monografie i prace zbiorowe poświęcone tej tematyce noszą tytuły, w których termin „filozofia" w odniesieniu do informatyki pojawia się najczęściej w połączeniach typu „informatyka a filozofia" lub też „filozofia i informatyka". Dotychczas jednak żaden z autorów nie zatytułował swojej monografii „Filozofia informatyki". Jaka może być tego przyczyna?

Informatyka jest dość „młodą" dyscypliną. Obecnie trudno jest odróżnić filozofię informatyki od filozofii umysłu, filozofii sztucznej inteligencji, filozofii informacji, filozofii obliczeń, filozofii techniki, a nawet filozofii matematyki. Zdarza się również, że autorzy stosują niektóre z tych pojęć zamiennie. I tak na przykład mówi się o filozofii informatyki i filozofii informacji, bądź też o filozofii informatyki i filozofii obliczeń. Twierdzi się nawet, że pojęcie „filozofia informatyki" jest zbędne, ponieważ taka dyscyplina nie istnieje; informatyka jest bowiem jedynie kolejnym działem inżynierii lub też jedną z wielu dyscyplin matematycznych, a zagadnienia filozoficzne z nią związane mieszczą się w rozważaniach filozofii techniki czy też filozofii matematyki. Czym zatem jest albo – czym powinna być filozofia informatyki?

Rapaport (2005a) pisze: „definiujemy 'filozofię X-a' jako badanie fundamentalnych założeń i głównych celów dyscypliny X" (s. 322). Zgodnie z tą definicją można uważać filozofię informatyki za rozważania dotyczące podstawowych założeń i celów informatyki. Wśród nich powinny się zatem znaleźć tematy ontologiczne, epistemologiczne i etyczne dotyczące tej dyscypliny.

W niniejszej książce przedstawiono główne kwestie związane z ontologią informatyki, a w szczególności rozważania dotyczące przedmiotu jej badań. Czym są obiekty zainteresowań informatyków? Jaka jest ich natura? Jakie istnieją między nimi związki?

Mówiąc o ontologii informatyki, należy rozpocząć od refleksji na temat samej tej nauki. Czy jednak mówiąc o informatyce, jesteśmy zgodni co do tego, o czym właściwie mówimy? Czy wiemy, co to jest informatyka i jaka jest jej istota? Co dokładnie mamy na myśli, kiedy mówimy: *informatyka* lub *computer science*?

Wydaje się, że odpowiedzi na to podstawowe pytanie już dawno udzielono, gdyż wytwory informatyki towarzyszą nam od dziesięcioleci. Wiemy zatem, jaką gałąź nauki nazywamy informatyką, choć nie zawsze rozumiemy, czym dokładnie zajmują się informatycy. Czy wiemy jednak cokolwiek o naturze tej dziedziny wiedzy i o tym, jakie jest jej miejsce wśród innych nauk?

To podstawowe pytanie ontologiczne – o tożsamość informatyki jako nauki – nie jest tylko problemem czysto filozoficznym, ma bowiem również swoje konsekwencje praktyczne. Pozwala bowiem odpowiadać na pytania: gdzie i w jaki sposób kształcić informatyków?, czy wydziały informatyki powinny znajdować się na uniwersytetach, obok wydziałów matematyki (lub nawet w ich obrębie), czy też na politechnikach i w innych szkołach technicznych?

Szukając odpowiedzi na pytanie, czym jest informatyka sama w sobie, warto przyjrzeć się pochodzeniu słów określających tę dyscyplinę. Polskie słowo *informatyka* zaproponował w 1968 roku Romuald Marczyński na odbywającej się w Zakopanem ogólnopolskiej konferencji poświęconej „maszynom matematycznym" (jak wtedy nazywano komputery). Pochodzenie tego słowa sugeruje bliski związek informatyki z informacją. Podobnie rzecz ma się w przypadku wyrazów określających tę dziedzinę nauki w języku francuskim (*informatique*) i niemieckim (*Informatik*). Jednakże angielska nazwa informatyki – *computer science* – nasuwa zupełnie inne skojarzenia. Budzi ona wiele kontrowersji i rodzi rozmaite pytania, nie tylko natury filozoficznej, lecz także praktycznej. Po pierwsze, pierwszy człon tej nazwy (*computer*) podkreśla związek definiowanej dyscypliny z komputerami, a nie z informacją (inaczej niż w języku polskim, francuskim czy niemieckim). Po drugie, słowo *science* sugeruje, że jest to dziedzina nauk ścisłych. Dlatego też wiele prac filozoficznych, szczególnie anglojęzycznych, poświęcono próbom odpowiedzi na pytanie, czy informatyka jest nauką ścisłą (*science*), czy też gałęzią inżynierii.

Do definiowania dyscypliny naukowej można podejść na dwa sposoby: poprzez zdefiniowanie obiektu jej badań bądź też – przez określenie metod i technik, jakimi się posługuje. W książce tej wybraliśmy pierwszą możliwość – próbując spojrzeć na informatykę przez pryzmat badanych przez nią obiektów. W każdym z czterech rozdziałów omówione zostały kwestie filozoficzne związane z jednym z obiektów najczęściej uważanych za przedmiot

badań informatyki: z komputerem, programem komputerowym, informacją oraz z rzeczywistością wirtualną.

W rozdziale pierwszym przeanalizowano problemy filozoficzne związane z twierdzeniem, że przedmiotem zainteresowań informatyków jest komputer, rozumiany dwojako: jako urządzenie oraz jako obiekt abstrakcyjny. Przedstawiono próby odpowiedzi na pytania związane z jego historią: która z maszyn liczących była pierwszym komputerem?, czy wynalazek ten został opatentowany?, a także – kogo można nazwać jego twórcą? Najwięcej miejsca poświęcono jednak kwestiom ontologicznym związanym z komputerami. Jakie cechy odróżniają komputer od innych maszyn? Czy jest to zawsze samodzielne urządzenie, czy też część innych, większych systemów? Często mówi się bowiem o komputerach wbudowanych w telefony, samochody, a nawet w sprzęt gospodarstwa domowego. Jakie cechy komputerów powinna uwzględniać ich ontologia? Czy możliwe jest zastosowanie do komputerów istniejących już ontologii artefaktów technicznych oraz jak rozwiązać związane z tym trudności, na przykład w jaki sposób określić ich funkcję i kryteria poprawnego działania. W rozdziale tym omówiono również konsekwencje definiowania komputera w terminach czysto syntaktycznych – jako obiektu matematycznego, gdzie nacisk położono na problem uniwersalnej realizowalności. Przedstawiono też inne, ciekawe z filozoficznego punktu widzenia charakterystyki komputera, w tym te, które oparte są na pojęciu obliczania.

Komputery, określane często jako „hardware", nieodłącznie związane są z wykonywanymi na nich programami – „software". Badanie związków pomiędzy terminami „hardware" i „software" oraz próby ich zdefiniowania jako odrębnych, jednoznacznie zaklasyfikowanych bytów są również jednym z tematów rozważań rozdziału pierwszego.

Programom komputerowym, rozumianym jako przedmiot badań informatyki, poświęcono rozdział drugi. Wielu informatyków twierdzi bowiem, że ich głównym zadaniem jest tworzenie programów i badanie ich własności, komputery natomiast interesują ich tylko jako maszyny, na których się je wykonuje. Czy program ma zatem pewnego rodzaju pierwszeństwo ontologiczne przed komputerami? Co to jest program i jakie są jego podstawowe cechy?

Pojęcie programu pojawia się w informatyce w różnych kontekstach, które przeanalizowano w rozdziale drugim. Dokonano w nim, między innymi, odróżnienia programu jako obiektu abstrakcyjnego od programu jako procesu fizycznego (wykonywanego na komputerze). Spróbowano również odpowiedzieć na pytanie, czy owa dualna natura programu – obiektu abstrakcyjnego i fizycznego jednocześnie – stawia w nowym świetle klasyczne problemy filozofii, takie jak związek pomiędzy umysłem a ciałem czy też

podział bytów na abstrakcyjne i konkretne. Czy programy komputerowe są przykładami obiektów o szczególnym statusie ontologicznym, które nie są ani konkretne, ani abstrakcyjne? Czym w istocie są programy komputerowe? Gdzie szukać ich ontologii – w filozofii muzyki czy też w filozofii matematyki? Jakie jest kryterium rozróżniania programów? Co o nich samych można wywnioskować z analizy procesu ich tworzenia?

Tworzenie programu nie jest prostą czynnością, lecz złożonym procesem, który rozpoczyna się sformułowaniem wymagań przed nim stawianych (stworzeniem specyfikacji), a kończy sprawdzeniem jego poprawności. W rozdziale drugim omówiono więc kwestie filozoficzne związane z różnymi rodzajami specyfikacji, próbując tym samym odpowiedzieć na pytania: jak rozróżnić specyfikacje od opisywanych przez nie programów?, czym są specyfikacje – definicjami warunkowymi artefaktów czy też odpowiednikami teorii naukowych? Przeanalizowano również wieloznaczny termin „implementacja", pojawiający się w informatyce w kontekście tworzenia programów.

Jednym z najszerzej dyskutowanych w literaturze zagadnień związanych z programami komputerowymi jest sprawdzanie ich poprawności. Jest to również historycznie jeden z pierwszych tematów refleksji filozoficznej związanej z informatyką, nie sposób więc go pominąć. W rozdziale drugim znalazły się zatem również rozważania związane z badaniem poprawności programów, mimo że wykraczają one poza ramy rozważań ściśle ontologicznych. Jak rozumieć termin „poprawny" w odniesieniu do programu komputerowego? Jakimi metodami badać „poprawność" programu? Czy tworzyć odpowiednie dowody formalne czy też ograniczyć się do testowania?

Współcześnie dominuje pogląd, że informatyka jest nauką o procesach przetwarzania informacji bądź też nauką o samej informacji. Czym zatem jest informacja? Jaka jest jej natura? Rozważania przedstawione w rozdziale trzecim są próbą odpowiedzi na pytania filozoficzne związane właśnie z informacją. Termin „informacja" jest bowiem używany w różnych kontekstach, zarówno w mowie potocznej, jak i w wielu dyscyplinach nauki – począwszy od informatyki, a skończywszy na naukach społecznych – często mówi się nawet o „nauce informacyjnej". Czy termin ten oznacza zawsze to samo? Czy informacja dla informatyka jest tym samym, czym informacja dla biologa zajmującego się genetyką? Jak rozumie się to pojęcie w informatyce? W rozdziale trzecim omówiono aspekty filozoficzne wybranych teorii informacji, kładąc szczególny nacisk na wykorzystywane w informatyce teorie matematyczne. Przedstawiono również próby odpowiedzi na pytania filozoficzne dotyczące natury informacji, wykraczające poza jej aspekt czysto ilościowy: czy informacja może być fałszywa?, jaki jest związek pomiędzy informacją, dezinformacją a informacją błędną?, co czyni informację

użyteczną?, jaki jest jej związek z wiedzą? Omówiono ponadto filozofię informacji: czym jest, jakimi zagadnieniami się zajmuje oraz jakie są jej podstawowe cele badawcze.

Rozdział czwarty natomiast poświęcono wirtualnej rzeczywistości, zjawisku związanemu z informatyką, które uważa się współcześnie – w dobie wszechobecnego Internetu oraz gier komputerowych – za jedno z podstawowych zainteresowań informatyków. W rozdziale tym przeanalizowano zagadnienia ontologiczne dotyczące wirtualnej rzeczywistości. Czym jest wirtualność i jak ją zdefiniować? Czy jest to nowe zjawisko, które pojawiło się wraz z rozwojem technologii komputerowych i które jest od nich całkowicie zależne? Jakie cechy ma wirtualna rzeczywistość? Jakim rodzajem rzeczywistości jest świat wirtualny? Jaki jest status ontologiczny jego obiektów? Co wiemy na temat związków wirtualności z realnością (światem rzeczywistym)? I wreszcie – czy pojawienie się wirtualnej rzeczywistości wpłynęło na zmiany pewnych klasycznych pojęć metafizyki?

Rozstrzygnięcia przedstawionych w niniejszej książce kwestii ontologicznych związanych z informatyką są istotne nie tylko dlatego, że pogłębiają nasze zrozumienie jej istoty, ale również z tego względu, że wpływają bezpośrednio na kwestie epistemologiczne i etyczne dotyczące tej stosunkowo młodej dziedziny wiedzy.

Dziękuję recenzentowi prof. Adamowi Olszewskiemu za cenne uwagi, które pozwoliły mi uniknąć pewnych błędów, oraz pani mgr Ewie Korek za ważne komentarze i korektę maszynopisu.

Składam wyrazy wdzięczności Ewie, Ani, Lilianie oraz Agacie, których wsparcie pomogło mi przetrwać trudne chwile mojego życia i dokończyć prace nad niniejszą książką.

Książkę tę dedykuję mojemu Mistrzowi, Profesorowi Romanowi Murawskiemu, który przez wiele lat, cierpliwie i z wielką życzliwością, wprowadzał mnie w świat filozofii. Bez Niego ta książka by nie powstała. Dziękuję.

Rozdział 1

Komputer

Wielu informatyków uważa, że informatyka jest nauką o komputerach, a ściślej rzecz ujmując, że jest dziedziną inżynierii zajmującą się tymi właśnie urządzeniami. Hamming (1969) pisze: „Sercem informatyki jest urządzenie technologiczne – komputer (*the computing machine*). Bez niego prawie cała nasza praca stałaby się jałową spekulacją. [...] Nadal wierzę, że ważne jest przyznanie, iż komputer, urządzenie przetwarzające informacje, jest fundamentem naszej dyscypliny" (s. 5).

Pogląd ten podzielają Newell, Perlis i Simon, którzy w roku 1967 zdefiniowali informatykę jako naukę o komputerach, podkreślając przy tym: „W definicji tej 'komputery' oznaczają 'żyjące komputery' – sprzęt, jego programy lub algorytmy i wszystko, co ich dotyczy. Informatyka jest nauką o zjawiskach związanych z komputerami" (s. 1374).

Jeśli jest zatem tak, że informatyka jest nauką o komputerach i powiązanych z nimi zjawiskach, to oczywiste staje się pytanie o to, czym właściwie komputer jest i jaką ma naturę.

W rozdziale tym przeanalizujemy zagadnienia filozoficzne związane z rozumieniem komputera jako urządzenia fizycznego. Ze względu na to, że określa się komputer także jako obiekt abstrakcyjny, rozważymy również trafność tego ujęcia. Omówimy trudności związane z rozróżnieniami pomiędzy terminem „hardware", najczęściej rozumianym jako sprzęt komputerowy, a „software". Przeanalizujemy także problemy filozoficzne związane z weryfikacją sprzętu komputerowego.

Przedstawmy więc różne sposoby definiowania komputera jako maszyny, szczególną uwagę zwracając na problematykę komputera jako specyficznego artefaktu.

1.1. Komputer jako urządzenie fizyczne

W rozumieniu potocznym komputery to fizyczne maszyny, które są wszechobecne – zarówno jako samodzielne urządzenia, jak i jako ich części: począwszy od telefonów poprzez samochody aż po promy kosmiczne. Warto zatem zastanowić się nad tym, które urządzenia spośród tych, z którymi spotykamy się na co dzień, są komputerami, a które jednak nie. Czy współczesne samochody lub telefony „naszpikowane" elektroniką są komputerami? Jakież to cechy musi posiadać urządzenie, by nazwać je komputerem?

Rozumienie terminu „komputer" zmieniało się nie tylko w zależności od czasu, ale również od tego, kto nazwą tą się posługiwał. W latach 80. XX wieku firma IBM utożsamiała komputer ze skrzynką na biurku, Intel zaś – z chipem wewnątrz tej skrzynki. Microsoft twierdził natomiast, że komputer jest to system operacyjny i oprogramowanie. W połowie lat 90. rozumiano komputer jako urządzenie, które można podłączyć do Internetu (Heyck, 2008). Zatem zdefiniowanie komputera nie było i nadal nie jest proste. Można sądzić, że próby precyzyjnego określenia terminu „komputer" wygenerowały więcej pytań niż odpowiedzi.

Czy komputer to chip w skrzynce na biurku, czy też system operacyjny i oprogramowanie? A może jest to konfiguracja sprzętu i oprogramowania w naszych urządzeniach? Sieć, do której są one podłączone? Jakie funkcjonalności powinien mieć komputer? Czy jest on kalkulatorem, urządzeniem sterującym czy też urządzeniem komunikacyjnym? Która z wielu powstałych w ciągu dziesięcioleci maszyn była pierwszym komputerem?

Omówienie powyższych zagadnień najtrafniej będzie rozpocząć od ich analizy w aspekcie historycznym. Rozpocznijmy zatem od próby odpowiedzi na ostatnie z powyższych pytań: Które urządzenie było pierwszym komputerem i kto był jego wynalazcą? W przypadku większości nowoczesnych wynalazków jesteśmy w stanie wskazać ich twórców, popularne przykłady to żarówka Edisona czy telefon Bella. W przypadku komputera jednak odpowiedź nie jest taka prosta, trudno jest bowiem jednoznacznie i ponad wszelką wątpliwość stwierdzić, czym właściwie komputer jest. Poglądy w tej kwestii są podzielone.

Jeśli więc przyjmiemy, że komputer jest tylko urządzeniem ułatwiającym proste obliczenia arytmetyczne, to za pierwsze komputery musimy uznać abaki i liczydła, a ich wynalazcy szukać trzeba wśród rachmistrzów starożytnych[1]. Jeśli z kolei przez komputer rozumiemy mechaniczne urządzenie liczące – wówczas jego pierwowzorem byłby „zegar liczący" zbudo-

[1] Wiele informacji dotyczących starożytnych abaków i liczydeł można znaleźć w książkach (Ifrah, 2006) i (Bondecka-Krzykowska, 2012a).

wany przez Wilhelma Schickarda w XVII wieku. Maszyna ta służyła bowiem do automatycznego dodawania i odejmowania, pomagała też mnożyć i dzielić. Pierwsze maszyny wykonujące w pełni automatycznie cztery podstawowe działania arytmetyczne powstały dopiero na początku XX wieku.

Czy definiując pojęcie komputera, powinniśmy brać pod uwagę tylko aspekt mechanizacji obliczeń?

Współczesne komputery są uniwersalne – wykonują nieporównanie większą klasę działań niż tylko obliczenia arytmetyczne. I tę właśnie uniwersalność traktuje się jako cechę odróżniającą je od innych maszyn. Szukając wynalazcy komputera, trzeba zatem odpowiedzieć na pytanie, komu zawdzięczamy ideę zbudowania maszyny do szerokich zastosowań.

Za twórcę koncepcji maszyny uniwersalnej uważa się Charlesa Babbage'a, który od roku 1833 prowadził prace nad projektem urządzenia nazwanego przez siebie „maszyną analityczną". W planach swojej maszyny Babbage przewidział nie tylko podstawowe elementy konstrukcyjne, których funkcjonalne odpowiedniki można znaleźć i we współczesnych komputerach (jednostkę liczącą, centralną jednostkę sterującą, pamięć, jednostkę wprowadzania danych i poleceń oraz jednostkę wyprowadzania wyników), ale również możliwość jej programowania. Był to więc pierwszy projekt „kalkulatora sterowanego programem". Maszyna Babbage'a, bez udziału człowieka, miała wykonywać sekwencje czynności – stała się więc pierwowzorem niecyklicznego automatu ze zmiennym programem umożliwiającym zmianę jej działania, niezależnym od wewnętrznej struktury[2]. Projektując ją, Babbage określił główne części i funkcje analitycznych maszyn obliczeniowych, przeznaczonych do rozwiązywania obszernej klasy problemów – a nie tylko jednego zadania. Dlatego właśnie maszynę analityczną nazywa się „dziadkiem współczesnych komputerów" (Ligonnière, 1992, s. 92).

Opisana maszyna miała być urządzeniem mechanicznym napędzanym silnikiem parowym. Jeśli więc przyjmiemy, że komputer jest urządzeniem elektrycznym, to maszyna Babbage'a nim nie była. Niełatwym zatem zadaniem okazuje się wskazanie pierwszego komputera.

Przez wiele lat uważano, że pierwszym komputerem elektronicznym był amerykański ENIAC, obecnie jednak wiadomo – między innymi dzięki odtajnieniu brytyjskich archiwów wojskowych – że do tego tytułu pretendują również brytyjskie maszyny Colossus oraz niemieckie Konrada Zuse.

ENIAC (Electronic Numerical Integrator And Computer) powstał jesienią 1946 roku, na Uniwersytecie Pensylwanii w Filadelfii, pod kierownictwem J.P. Eckerta i J.W. Mauchly'ego, dla potrzeb wojskowych. Podobnie jak

[2] Niestety, za życia konstruktora projekt maszyny nie doczekał się realizacji.

Colossus, nie posiadał on automatycznego sterowania, wykonywał ciąg instrukcji niezależnie od wprowadzonych danych. Wszystkie decyzje dotyczące ewentualnych zmian działania urządzenia musiał podejmować człowiek. W wyniku wielu dyskusji nad ulepszeniem ENIAC-a, prowadzonych na Uniwersytecie Pensylwanii, powstała koncepcja „programu wprowadzanego". Polegała ona na wprowadzeniu instrukcji do wewnętrznej pamięci maszyny, dzięki czemu mogła ona przetwarzać polecenia programu – tak samo jak dane. Zakładano, że zwiększy to „elastyczność" urządzenia.

W roku 1944 powstał plan zbudowania maszyny EDVAC (Electronic Discrete Variable Automatic Computer). Choć została ona skonstruowana i uruchomiona w Cambridge przez zespół Johna von Neumanna dopiero w 1952 roku[3], to jednak wcześniej, 30 czerwca 1945 roku uczony ten opublikował tekst pt. „First Draft of Report on EDVAC". Opisał w nim między innymi koncepcję programu wprowadzanego. Było to pierwsze naukowe opracowanie prezentujące spójne teoretyczne podstawy automatów z takim programem. Dzięki niemu pojęcie „komputer" zyskało rangę naukową. Sam John von Neumann nigdy jednak nie zgłaszał, w żadnej formie, pretensji do autorstwa wynalazku.

Inaczej rzecz się miała w przypadku twórców ENIAC-a. Eckert i Mauchly złożyli stosowny wniosek patentowy – tym samym rozpoczynając wieloletni spór o to, kto jest wynalazcą komputera. Dopiero w roku 1973 sąd w Minneapolis spór ten rozstrzygnął, uznając prawa Eckerta i Mauchly'ego co do konstrukcji ENIAC-a, odmawiając natomiast przyznania im praw autorskich do samego wynalazku. Za twórcę komputera sąd uznał Johna Vincenta Atanasoffa, który w latach 30. XX wieku zbudował maszynę ABC[4].

Obecnie inżynierowie pod pojęciem „komputer" rozumieją maszynę ogólnego przeznaczenia, wykorzystywaną do przetwarzania informacji lub symboli. Ifrah (2006) podaje następującą definicję:

> Komputer jest automatem złożonym z urządzenia wejściowego i wyjściowego, pamięci, jednostki centralnej dokonującej wszelkiego rodzaju przekształceń danych wyrażonych w postaci ciągu znaków (stanowiących materialne przedstawienie zakodowanych informacji), który pozwala – w granicach fizycznych możliwości urządzenia – wykonywać wszelkie obliczenia typu symbolicznego (a zatem rozwiązywać wszelkie zadania, których rozwiązania można wyrazić w postaci algorytmu) pod kierunkiem jednostki sterującej, działającej zgodnie z programami zapisanymi w pamięci (i wobec tego traktującej instrukcje wykonawcze w ten sam sposób jak dane przeznaczone do przetworzenia) (s. 846).

---

[3] Z powodu późnego ukończenia prac czasami podaje się w wątpliwość, czy EDVAC był pierwszym komputerem z wprowadzanym programem.

[4] ABC służył do rozwiązywania równań różniczkowych. Nie była to maszyna w pełni automatyczna ani programowalna.

Powyższy opis uwzględnia zarówno uniwersalność maszyny i jej podstawowe elementy zaprojektowane przez Charlesa Babbage'a, jak i koncepcję „programu wprowadzanego" opisaną przez von Neumanna. Są to składowe tak zwanej architektury von Neumanna – podstawowej architektury współczesnych komputerów[5].

Dość często wśród cech definiujących komputer wymienia się również tę, że komputer jest urządzeniem cyfrowym. Jednakże nie wszystkie maszyny nazywane komputerami były i są cyfrowe – istnieją również wersje analogowe i hybrydowe (zawierające komponenty obu tych rodzajów). Co więcej, także interpretacja terminu „cyfrowe" w odniesieniu do komputerów jako urządzeń fizycznych niesie ze sobą kilka problemów.

Po pierwsze, większość – jeśli nie wszystkie – systemów fizycznych uważanych za komputery można interpretować zarówno w terminach cyfrowych, jak i analogowych. Rozważmy przykład wspomnianej wcześniej maszyny liczącej Schickarda: Czy jest to maszyna cyfrowa czy analogowa? Jeśli interpretujemy tryby w kołach zębatych jako cyfry, a pełne obroty tych kół jako zmiany stanów, to jest to maszyna cyfrowa. Z drugiej jednak strony, jeśli rozumiemy koła zębate jako reprezentujące continuum wartości i skoncentrujemy się na ich ciągłych obrotach, to „zegar liczący" jest komputerem analogowym.

Po drugie, istnieją trudności z jednoznacznym zaklasyfikowaniem konkretnych urządzeń liczących jako analogowych bądź cyfrowych. Prostym tego przykładem jest suwak logarytmiczny, uważany najczęściej za urządzenie analogowe. Dokładność jego jest jednak ograniczona (w zależności od możliwości percepcyjnych konkretnego użytkownika) do dwóch lub trzech miejsc po przecinku.

A zatem opisy fizycznych maszyn w terminach cyfrowe/analogowe mogą być mylące. Alan Turing (1950) pisze:

> [maszyny dyskretnie stanowe] to takie, które działają przez nagłe przeskoki lub kliknięcia z jednego całkiem oznaczonego stanu do innego. Stany te są dostatecznie różne, by zignorować możliwość ich pomylenia. Ściśle mówiąc, nie ma takich maszyn. Wszystko w rzeczywistości porusza się w sposób ciągły. Ale istnieje wiele rodzajów maszyn, o których można z pożytkiem myśleć jak o maszynach będących maszynami dyskretnie stanowymi (s. 439).

Bardziej właściwe wydaje się zatem rozumienie rozróżnienia cyfrowy/analogowy na poziomie symbolicznym, a nie – jako opisu konkretnych maszyn. Przekonanie o istnieniu pewnych istotnych (wewnętrznych) cech

---

[5] Warto wspomnieć, że nie wszystkie współczesne komputery są realizacjami tej architektury.

komputerów, dzielących je na analogowe i cyfrowe, to jeden z mitów informatyki (Moore, 1978), który może prowadzić do stawiania pytań takich jak: Czy mózg jest komputerem cyfrowym czy analogowym?

Jakie zatem istotne cechy mają wszystkie maszyny nazywane komputerami i jaki jest ich status ontologiczny? Rozpocznijmy od omówienia ontologii komputerów jako artefaktów technicznych.

1.1.1. Komputer jako artefakt

Komputer jest dziełem człowieka, które służy realizacji pewnych celów, określonych przez jego projektantów i użytkowników. Obiekty powstałe w wyniku celowych ludzkich poczynań, w odróżnieniu od tych stworzonych przez naturę lub powstałych w wyniku przypadkowych aktów człowieka, nazywa się artefaktami.

Przyjrzyjmy się, czym jest artefakt z punktu widzenia filozofii.

Stanford Encyclopedia of Philosophy podaje, że artefakt to „obiekt, który został stworzony lub wyprodukowany w pewnym celu" (Hilpinen, 2011). Wyraz ten pochodzi od łacińskich słów: *arte*, formy gramatycznej słowa *ars* oznaczającego sztukę, oraz *factum*, słowa pochodzącego od czasownika *facere* – wytwarzać, co dobrze oddaje jego znaczenie. Najbardziej ogólne rozumienie słowa „artefakt" wyraża się w definicji: „coś, co jest dziełem ludzkiego umysłu i pracy, w odróżnieniu od wytworów natury"[6].

Artefakty to nie tylko narzędzia, urządzenia czy inne obiekty fizyczne, ale również pojęcia bardziej abstrakcyjne, takie jak: kodeksy prawne, pieniądze, sztuki teatralne oraz utwory muzyczne. Najciekawszą z punktu widzenia filozofii informatyki kategorią artefaktów są tak zwane artefakty techniczne – albowiem ich przykładami są właśnie urządzenia nazywane komputerami. Kroes i Meijers (2006) piszą:

> Artefakty techniczne, takie jak maszyny do pisania, młotki, kopiarki lub komputery, różnią się od artefaktów społecznych – takich jak prawa lub pieniądze – tym, że realizacja ich funkcji w zasadniczym stopniu zależy od struktury fizycznej. Różnią się również od obiektów fizycznych czy naturalnych, ponieważ są wyprodukowane celowo (intencjonalnie) i używane przez istoty ludzkie do realizacji pewnych celów (s. 1).

Tak określone artefakty techniczne mają dwie zasadnicze cechy: 1) są obiektami fizycznymi, których własności decydują o realizacji funkcji oraz 2) zostały wyprodukowane do realizacji pewnych celów. Można zatem powiedzieć, że mają one naturę dualną.

---

[6] Słownik języka polskiego PWN, http://sjp.pwn.pl/szukaj/artefakt.

W literaturze filozoficznej powszechne wydaje się przekonanie o takiej właśnie, tj. dualnej, naturze artefaktów technicznych (Baker, 2006; Houkes, Meijers, 2006; Kroes, Meijers, 2006; Kroes, 2010). Z jednej strony są one fizyczne, ich struktura daje możliwość realizacji pewnych funkcji, z drugiej zaś strony funkcje te odnoszą się do czegoś niematerialnego – ludzkich zamiarów (intencji). Zauważmy, że takie określenie artefaktów technicznych – czyli jako obiektów fizycznych i intencjonalnych zarazem, łączy dwa różne sposoby patrzenia na świat. Z jednej strony, można traktować świat jako złożony z obiektów fizycznych połączonych pewnymi związkami. Jednakże można też patrzeć na rzeczywistość poprzez pryzmat agentów, przede wszystkim ludzi, przedstawiających świat intencjonalnie i celowo w nim działających. Artefakty techniczne jako obiekty fizyczne przynależą do pierwszej wizji świata, natomiast jako obiekty posiadające intencjonalnie nadane funkcje mieszczą się w wizji drugiej. Zatem aby scharakteryzować omawiane artefakty, niezbędne są obie konceptualizacje świata. Kroes i Meijers (2006) twierdzą, że „[t]o czyni artefakty techniczne obiektami 'hybrydowymi', które mogą być opisane właściwie w sposób, który łączy fizyczne i intencjonalne konceptualizacje świata" (s. 2).

Do dziś jednak nie powstała żadna powszechnie akceptowana teoria wymienionych artefaktów, która łączyłaby oba nieodłączne ich aspekty: fizyczny i intencjonalny (funkcjonalny). Największą trudnością istniejących teorii wydaje się być zdefiniowanie pojęcia funkcji artefaktów. Albowiem z jednej strony, jeśli rozumie się funkcję jako możliwą do zrealizowania w obiektach fizycznych, powstaje pytanie, w jaki sposób funkcja ta związana jest ze stanami umysłu ludzkiego, tworzącymi rdzeń konceptualizacji intencjonalnej. Jeśli jednak przyjmiemy, że funkcja jest pewnym stanem umysłu, to istnieje ona tylko w wyobraźni projektantów i użytkowników artefaktów. Trudno jest wówczas wyjaśnić, w jaki sposób jest ona związana z substancją fizyczną tworzącą konkretny artefakt. Tak więc pojęcie funkcji artefaktu jest pojęciem kluczowym dla jego opisu, łączącym dziedzinę fizyczną i intencjonalną (Kroes, Meijers, 2006).

W literaturze dotyczącej artefaktów technicznych (por. np. Kroes, 2010; Meijers, 2001; Thomasson, 2007; Vermaas i Houkes, 2003) znaleźć można informacje na temat dwóch podstawowych koncepcji ich funkcji: teorie przyczynowe (ang. *causal role theories*) oraz intencjonalne.

Te pierwsze zakładają, że aktualne możliwości fizyczne wyznaczają funkcję. Jednym z przykładów jest analiza funkcjonalna Cumminsa (1975). Cummins twierdził, że nie istnieje artefakt bez obiektu fizycznego i jego aktualnych własności, natomiast wszystkie stwierdzenia funkcyjne mogą być traktowane jako części „analizy funkcjonalnej". Pisze on: „*x* funkcjonuje jako φ w *s* (lub: funkcją *x* – *a* w *s* jest φ) względem analitycznego wyjaśnie-

nia *A* zdolności *s* – *a* do ψ tylko w przypadku, gdy *x* jest zdolny do φ w *s* oraz *A* właściwie i wystarczająco tłumaczy zdolność *s* – *a* do ψ przez, w części, odwołanie do zdolności *x* – *a* do φ w *s*" (s. 762).

Przykład: funkcją serca jest pompowanie krwi, gdyż – jak to wyjaśnia fizjologia – jest to jego przyczynowy wkład w aktywność układu krążenia. Funkcją wskazówki godzinowej zegara natomiast jest pokazywanie godzin, ponieważ taki jest mechanizm działania zegara. A zatem funkcjami są te aktualne aktywności obiektu, które są przyczynowymi wkładami w aktywność szerszego układu, wskazywanego przez pewne analityczne wyjaśnienie.

Teoria taka ma dwie wady. Jest ona zbyt liberalna, gdyż nie nakłada wystarczających ograniczeń na funkcję – i tak, pozwala stwierdzić, że powodowanie raka skóry jest funkcją słońca, ponieważ istnieje opisany analitycznie wpływ słońca na powstawanie nowotworowych zmian skórnych. Z drugiej jednak strony uznać ją można za zbyt wąską, bo koncentruje się jedynie na aktualnych możliwościach fizycznych przedmiotu. Struktura i funkcja stają się wówczas niemal identyczne. Ostatecznie jednak przyjąć można, że największą wadą tego ujęcia jest brak kryteriów określających poprawne działanie artefaktu.

Sformułowanie „poprawne działanie artefaktu" rozumie się powszechnie jako działanie zgodne z jego funkcją, wyrażane często stwierdzeniami typu: „urządzenie robi to, co powinno" – czyli to, do czego je zaprojektowano. Czy można określić „poprawne działanie" artefaktu bezpośrednim odwołaniem się tylko do jego właściwości fizycznych? Wtedy urządzenie, które nie ma obecnie fizycznych zdolności do wykonywania zamierzonej funkcji – z definicji funkcji tej nie posiada. A więc nie można mówić o złym funkcjonowaniu samolotu, który (na przykład z powodu błędów konstrukcyjnych) nie lata, ponieważ zgodnie z teorią przyczynową latanie nie jest funkcją tego samolotu. Intencje związane z artefaktem stają się wtedy, w sposób zupełnie nieintuicyjny, nieistotne dla określenia jego funkcji.

W jaki sposób można zatem określić, czy urządzenie działa poprawnie, czy też nie? Konieczne jest odwołanie się do intencji jego projektanta. W wielu wypadkach działanie artefaktu, jakim jest komputer, jego użytkownik uznać może za niepoprawne – na przykład wówczas, gdy urządzenie nie uruchamia się lub działa wolniej niż zazwyczaj. Jego działanie może być jednak całkowicie poprawne z punktu widzenia projektanta, ponieważ komputer aktualizuje w tym czasie, w sposób automatyczny (na przykład korzystając z sieci), swoje oprogramowanie. A zatem każde działanie urządzenia może zostać uznane za poprawne lub wadliwe w zależności od intencji jego projektanta. To, czy maszyna działa poprawnie, nie jest własnością samej maszyny jako obiektu fizycznego, lecz zostało wyznaczone

w terminach planu (specyfikacji) określonego przez jej twórcę. Artefakt fizyczny nie może przecież sam ustalać własnego kryterium poprawności[7]. Jest to najpoważniejszy argument przeciw przyczynowym teoriom funkcji.

Podobnych trudności, związanych z kryterium poprawnego działania artefaktu, nie napotykamy w teoriach intencjonalnych (por. np. McLaughlin, 2001; Searle, 1995), bowiem zakłada się w nich, że to agent przypisuje funkcję do artefaktu, a obiekt i jego elementy pełnią określoną funkcję tylko wtedy, gdy realizują jego cel. McLaughlin (2001) pisze: „Funkcja artefaktu jest pochodną celu jakiegoś agenta, jaki miał on, tworząc lub przystosowując obiekt; jest ona nadawana obiektowi przez jego oczekiwania i przekonania. Nie ma agenta, nie ma celu, nie ma funkcji" (s. 60).

Trzeba więc w tym miejscu postawić pytanie, w jaki sposób funkcja jest „przydzielana" obiektowi. Można przyjąć, że jest ona wyznaczana przez świadomość projektantów i użytkowników artefaktów. Trudno jest wówczas uzasadnić, jak ludzkie myślenie może ograniczać rzeczywiste przedmioty, takie jak artefakty. Zatem jeśli funkcje owe traktowane są pierwotnie jako wzorce stanów mentalnych, istniejące tylko w ideach projektantów i użytkowników artefaktów, to niejasny jest stosunek funkcji do fizycznej substancji w konkretnym obiekcie (Kroes, 2010). Skoro pojęcie funkcji artefaktu jest nieodłącznie związane ze stanami świadomości agentów, to ich mózgi są obiektami fizycznymi, w których pozostaje ona zlokalizowana. Taka wersja teorii intencjonalnej jest więc szczególnym przypadkiem teorii przyczynowej, w której „mózg agenta" jest nośnikiem funkcji artefaktu (Turner, 2013a).

Istnieją również inne, alternatywne teorie funkcji artefaktów. Można tu wspomnieć Vermaasa i Houkes'a (2003) – choć nie stworzyli oni całościowej teorii funkcji dla artefaktów technicznych, to zaproponowali jednak cztery postulaty, które każda taka teoria powinna spełniać.

> D1. *Właściwe vs przypadkowe*: Teoria funkcji powinna rozróżniać pomiędzy właściwymi i przypadkowymi funkcjami artefaktu. Funkcje właściwe winny być rozumiane jako funkcje przypisane standardowo do artefaktu, podczas gdy funkcje przypadkowe bywają im nadawane tylko okazjonalnie.
> D2. *Wadliwe działanie*: Teoria funkcji powinna umożliwiać przypisanie właściwych funkcji do artefaktów działających wadliwie – to znaczy takich, które nie są w stanie funkcji tych spełniać.
> D3. *Struktura fizyczna:* Teoria funkcji powinna wymagać, by dla każdej funkcji istniały warunki strukturalne wystarczające do jej spełnienia. […]
> D4. *Nowość*: Teoria funkcji powinna umożliwiać określenie właściwej funkcji artefaktów nowatorskich bądź nietypowych (s. 265-266).

[7] Do zagadnień związanych z weryfikacją urządzeń powrócimy w paragrafie 1.1.3.

Za przedstawionymi powyżej postulatami kryją się proste intuicje. I tak, D1 zwraca uwagę na możliwość odróżnienia sytuacji typowego zastosowania artefaktów od przypadkowego. Drukarka komputerowa służy zazwyczaj do drukowania, i choć można użyć jej jako stołka (zapewne nie bez trudu), by sięgnąć po książkę z najwyższej półki na regale, to jednak funkcją drukarki jest drukowanie, a nie inne, przypadkowe (okazjonalne) jej zastosowanie. Postulat D2 pozwala na wyeliminowanie zastrzeżeń stawianych pod adresem teorii przyczynowych – w tym teorii Cumminsa – którym brak kryterium określającego wadliwe funkcjonowanie artefaktów. Warunek D3 uniemożliwia określanie funkcji dla obiektów, które nie posiadają struktury fizycznej odpowiedniej dla jej realizacji. Obrazuje to przykład kulki z plasteliny, której funkcją nie może być wbijanie gwoździ, gdyż jest ona zbyt miękka. Młotek natomiast ma właściwą strukturę fizyczną do realizacji tego zadania. Ostatni postulat (D4) gwarantuje możliwość odpowiedniego przypisania nowego zadania wynalazkom – na przykład powstałym przez połączenie istniejących artefaktów – gdy takie połączenie ma funkcję inną niż obiekty, z których ów wynalazek powstał.

Nietrudno zauważyć, że opisana wcześniej teoria Cumminsa nie spełnia przynajmniej dwóch z przedstawionych powyżej postulatów, a mianowicie *wadliwe działanie* oraz *właściwy vs przypadkowy*. Jeśli aktualne możliwości fizyczne wyznaczają funkcję obiektu, to nie tylko nie można określić obiektywnego kryterium jego wadliwego działania, ale również nie sposób odróżnić jego funkcji właściwej od przypadkowej.

Oprócz teorii przyczynowych i intencjonalnych, w literaturze znaleźć można próby zastosowania innych teorii funkcji do artefaktów technicznych. Vermaas i Houkes (2003), po przeanalizowaniu możliwości wykorzystania etiologicznych teorii funkcji znajdujących zastosowanie w biologii do artefaktów technicznych, dochodzą do wniosku, że żadnej z nich nie można zastosować wprost do tych obiektów. Teorie etiologiczne bowiem przypisują obiektom funkcje, bazując na ich historii przyczynowej – a wiemy, że nie ma dla nich dobrze rozwiniętej i powszechnie akceptowanej teorii przyczynowej. Vermaas i Houkes (2003) zrobili jednak pierwszy krok ku stworzeniu etiologicznej teorii artefaktów technicznych, nadającej obiektom funkcje na podstawie historii ich projektowania, podając jej trzy podstawowe założenia: „(1) autorzy projektują artefakt, który ma pełnić pewną funkcję, (2) wyznaczają następnie jego fizyczną strukturę – bazując na swej wiedzy naukowej i technicznej, oraz (3) wyjaśniają, dlaczego artefakt z tak określoną strukturą fizyczną może pełnić ową funkcję" (s. 287). Tak rozumiana teoria funkcji spełnia wszystkie przedstawione wcześniej postulaty.

I tak, spełnia ona warunek *struktury fizycznej* – poprzez wykorzystanie wiedzy naukowej i technicznej projektantów, gwarantującej, że struktura

fizyczna artefaktu umożliwi realizację stawianych przed nim funkcji. Pozwala również na zdefiniowanie wadliwego działania artefaktu w przypadku, gdy jego aktualna struktura fizyczna jest niewłaściwa, czyli niezgodna z wiedzą projektantów. Do sytuacji takiej dochodzi, gdy artefakt jest wadliwie wyprodukowany. A zatem teoria ta spełnia wymóg D2. Jednak propozycja przedstawiona przez Vermaasa i Houkesa wymaga doprecyzowania, na co zwracają uwagę sami jej autorzy, na przykład poprzez zastąpienie odwołania się wprost do intencji twórców odniesieniem do planów czy projektów.

Przedstawione powyżej trudności związane ze sformułowaniem odpowiedniej dla artefaktów technicznych teorii funkcji pokazują, że stworzenie ontologii tych artefaktów nie jest zadaniem łatwym. Houkes i Meijers (2006) nazywają je wprost „trudnym problemem”[8]. Podane przez nich dwa kryteria adekwatności ontologii do artefaktów technicznych – niedookreślenie i ograniczenia realizowalności – odpowiadają w pewnym sensie ich dualnej naturze.

Niedookreślenie: Ontologia artefaktów powinna dopuszczać sytuację, kiedy pojedyncza funkcja może być spełniana w różnych strukturach oraz gdy jedna struktura może spełniać różne funkcje. Relacja pomiędzy artefaktem i jego funkcją nie musi być bowiem relacją jeden do jeden.

Ograniczenia realizowalności: Powinna ona także umożliwiać wyciąganie wniosków na temat materialnej bazy z twierdzeń na temat artefaktów i ich funkcji i *vice versa* – powinna dawać wgląd w interakcje pomiędzy materialną podstawą a artefaktem. Informacja o artefakcie niesie pewną informację o jego materialnej bazie – i istnieje też zależność odwrotna.

Większość badaczy zgadza się z tym, że pierwsze z powyższych kryteriów powinna spełniać każda ontologia artefaktów technicznych. Jednakże w przypadku drugiego kryterium zdania są podzielone. Baker (2006) pisze, że związek pomiędzy artefaktem i jego materialną bazą: „[…] zależy od specyfiki konkretnej funkcji: mając daną określoną funkcję – powiedzmy obieranie marchewki – pytamy, jakiego rodzaju materiału użyć i jaki powinien on mieć kształt”(s. 134).

Houkes i Meijers (2006) analizują dwa sposoby rozumienia relacji pomiędzy artefaktem i jego materialną bazą: superweniencję oraz ukonstytuowanie, twierdząc, że żaden z nich podanych powyżej kryteriów nie spełnia. Rozważmy zagadnienie superweniencji.

Pojęcie to jest jednym z centralnych pojęć filozofii umysłu służącym do wyjaśniania związków pomiędzy tym, co fizyczne, i tym, co mentalne.

---

[8] Houkes i Meijers zatytułowali swój artykuł z 2006 roku *The Ontology of the Artefacts: the Hard Problem*.

Twierdzi się, że procesy mentalne superweniują (są „nadbudowane") na stanach mózgu. Z superweniencją wiązane są trzy główne założenia: (1) jeśli dwa obiekty są nierozróżnialne ze względu na swoje własności bazowe, to muszą być one nierozróżnialne również ze względu na własności nadbudowane, (2) własności nadbudowane są zależne od własności bazowych (fizycznych, chemicznych, biologicznych i tym podobnych), przy czym w przypadku własności mentalnych można je realizować w strukturach fizycznych na wiele sposobów, (3) własności nadbudowane są nieredukowalne do własności bazowych. Podstawowa idea związana z superwenienciją zawiera się w twierdzeniu, że żadne dwie rzeczy jednakowe fizycznie nie różnią się między sobą także na poziomie mentalnym.

Takie ujęcie związku fizyczne/umysłowe nie spełnia obu przytoczonych wyżej kryteriów dla ontologii artefaktów technicznych. Kryterium niedookreśloności jest złożeniem dwóch założeń: dana funkcja artefaktu może być fizycznie realizowana w wielu strukturach fizycznych (*top-down*) oraz dana struktura fizyczna może pełnić wiele funkcji (*bottom-up*). Oczywiście, kryterium *top-down* zgadza się z ideą superweniencji, jednak problem pojawia się w przypadku drugiego kryterium, gdyż dwa przedmioty o tej samej strukturze fizycznej nie mogą być rozróżnialne na poziomie funkcjonalnym. Zatem dwa identyczne przedmioty nie mogą pełnić różnych funkcji.

Próbą rozwiązania powyższej kwestii jest rozszerzenie zbioru własności bazowych i rozważanie artefaktów nie w izolacji, ale razem z kontekstem ich projektowania i użycia. Można wówczas stwierdzić, że w jednym kontekście artefakt (na przykład wspomniana wcześniej drukarka) pełni funkcję F1 (drukuje), ale już w innym – pełni funkcję F2 (ułatwia zdejmowanie książki z półki). Podobne ujęcie tego zagadnienia wymaga od nas jednak dalszych rozważań. Po pierwsze, nie można odpowiednio rozszerzyć bazy superweniencji bez odwołania się do funkcji artefaktu – która to funkcja jest z kolei nadbudowana na własnościach bazowych, ponieważ prowadzi to do tak zwanego błędnego koła. Po drugie, artefakty mają nieodłączne własności mentalne, są one bowiem projektowane, produkowane i używane jako obiekty przez intencjonalnych agentów. Zatem baza powinna zawierać zarówno własności fizyczne, jak i mentalne – co przeczy samej idei superweniencji. Jak widać, idea rozszerzenia zbioru własności bazowych nie rozwiązuje problemu.

Superweniencja nie spełnia również drugiego kryterium dla ontologii artefaktów technicznych – ograniczenia realizowalności. Artefakty takie muszą mieć bowiem określoną strukturę fizyczną, by móc realizować swoją funkcję. Jest to rozumowanie typu: od funkcji do struktury fizycznej, czyli od własności mentalnej do fizycznej bazy. Idea superweniencji dopuszcza jednak tylko rozumowanie odwrotne, od bazy fizycznej do własności mental-

nych (różne obiekty mogą pełnić tę samą funkcję, ale nie na odwrót). Tak więc dla ontologii artefaktów fizycznych superweniencja nie jest dobrą teorią opisującą związki pomiędzy tym, co fizyczne, a tym, co mentalne.

Ontologię alternatywną zaproponowała Lynne R. Baker (2000). Jej zdaniem, spoiwem świata materialnego jest ukonstytuowanie (ang. *constitution*). Pisze ona: „Każdy obiekt ma swój zasadniczy, pierwotny (podstawowy) rodzaj, a byty różnych pierwotnych rodzajów mają odmienne warunki trwania. Ukonstytuowanie jest relacją pomiędzy rzeczami o różnych rodzajach pierwotnych" (s. 100). Relacja ukonstytuowania jest relacją asymetryczną i niezwrotną, która zachodzi pomiędzy obiektami o różnych rodzajach pierwotnych, ale o tym samym zbiorze własności, w konkretnych okolicznościach powodując powstanie nowego obiektu. Baker definiuje przy tym dwa rodzaje własności obiektów: podstawowe (ang. *essential*) i pochodne (ang. *derivative*). Na przykład, jeśli mechanizm zegarka wytwarza dźwięk tykania, to zegarek tyka – tykanie jest więc podstawową własnością mechanizmu, ale pochodną własnością zegarka. Ważną rolę w teorii Baker odgrywa również pojęcie okoliczności. Suma molekuł wody – sama w sobie – nie tworzy obiektu o innym rodzaju pierwotnym, na przykład rzeki, lecz czynią to odpowiednie okoliczności.

Analizując rozumowanie autorki, przyjrzyjmy się teraz temu, w jaki sposób można zastosować powyższą teorię do artefaktów technicznych.

Baker charakteryzuje ich materialną bazę jako agregat – całość złożoną z elementów. I tak, materialną bazą młotka jest suma drewna i stali. Obiekt ma przy tym pewien nowy, w stosunku do swoich części, rodzaj pierwotny – funkcję, którą określa jego projektant lub producent. Autorka definiuje obiekt *x* jako artefakt wtedy i tylko wtedy, gdy: (1) ma on jednego lub więcej projektantów *d*, których intencje częściowo wyznaczają jego właściwą funkcję i od których zależy istnienie *x*, (2) *x* jest ukonstytuowany przez agregat, wybrany lub ustalony przez *d* do realizacji funkcji dla niego właściwej. Powracając do dualnej natury artefaktów, można stwierdzić, że (1) oddaje jego aspekt intencjonalny, a (2) naturę fizyczną.

Pozostaje jeszcze określenie okoliczności, w jakich agregat *a* konstytuuje artefakt *x*. Agregat *a* jest odpowiedni wtedy i tylko wtedy, gdy: (1) *a* zawiera dostatecznie dużo elementów o odpowiedniej strukturze, by umożliwić artefaktowi *x* spełnianie jego właściwej funkcji, (2) elementy *a* można połączyć w całość w taki sposób, by artefakt mógł funkcję tę spełniać. Oczywiście w praktyce bardzo trudno jest określić w sposób dokładny i wyczerpujący, w jakich okolicznościach agregat *a* konstytuuje artefakt *x*.

Podstawowym pojęciem teorii Baker jest ukonstytuowanie. Houkes i Meijers (2006) definiują je następująco: „*a* konstytuuje *x* w czasie *t* wtedy

i tylko wtedy, gdy *a* oraz *x* są przestrzennie zbieżne w czasie *t* w *x*-sprzyjających okolicznościach" (s. 126).

W teorii ukonstytuowania funkcja właściwa artefaktu determinuje jego rodzaj właściwy. Taki esencjalizm[9] sprawia jednak trudności w czytelnym wyjaśnieniu pewnych kwestii. Istnieją artefakty – jak na przykład telefony komórkowe czy komputery – które są wielofunkcyjne, to znaczy mają wiele różnych funkcji właściwych. Co więcej, niektóre obiekty zyskują z czasem nowe funkcje, bez utraty pierwotnej. Dobrym tego przykładem jest lek o nazwie aspiryna. Początkowo był on stosowany jako lek przeciwbólowy i przeciwzapalny, jednak gdy okazało się, że ma także własności rozrzedzające krew, zastosowano go do leczenia chorób układu krążenia. Aspiryna uzyskała zatem nową funkcję właściwą, nieeliminującą tej poprzedniej (nadal działa ona przeciwbólowo). Jak określić taki artefakt, korzystając z teorii ukonstytuowania, w myśl której funkcja właściwa *x* pomaga wyznaczyć tożsamość *x*?

Jeśli istnieją dwie funkcje właściwe – mamy do czynienia z dwoma przestrzennie zbieżnymi artefaktami. Przypuśćmy, że ktoś kupuje opakowanie aspiryny jako leku rozrzedzającego krew. Będąc na spotkaniu z osobą, którą rozbolała głowa, częstuje ją tabletką z opakowania – czyli lekiem przeciwbólowym. Czy w opakowaniu znajdują się wówczas dwa różne artefakty, skoro tabletki pełnią odmienną funkcję? Jaki jest związek pomiędzy nimi? Zjawisko to Houkes i Meijers (2006) nazywają „*ontological stacking*, czyli istnieniem wielorakich zbieżnych przestrzennie artefaktów" (s. 126).

Zjawisko zawiera w sobie pewną trudność związaną nie tylko z teorią ukonstytuowania, lecz także z powszechną tendencją identyfikowania funkcji właściwej z esencją artefaktu oraz ograniczaniem pojęcia projektowania jedynie do skonstruowania produktu lub jego projektu. Jednym z możliwych rozwiązań tej kwestii, zaproponowanym przez Houkesa i Vermaasa (2004), jest rozszerzenie pojęcia projektowania artefaktu o jego konstruowanie oraz ogłoszenie sposobu używania. Aspiryna jest wówczas lekiem przeciwbólowym, jeśli zostanie użyta do ukojenia bólu. Z kolei Baker (2006) proponuje następujące rozwiązanie problemu *ontological stacking*. Pisze ona o aspirynie, jako leku rozrzedzającym krew, w następujący sposób: „[…] ontologicznie mówiąc, mamy teraz nieznacznie inny artefakt – aspirynę* – która również jest nazywana 'aspiryną', jednakże to aspiryna* ma istotę wielofunkcyjną: przeciwbólowo-rozrzedzającą krew" (s. 135).

Można łatwo uzasadnić, że aspiryna i aspiryna* to dwa różne obiekty. Jeśli aspiryna przestałaby działać przeciwbólowo, to zaprzestano by jej

---

[9] Przez esencjalizm rozumiemy tu stwierdzenie, że można zdefiniować obiekt przez podanie skończonego zbioru cech (*esencji*), które obiekt ten musi posiadać, by należeć do definiowanej grupy.

produkcji. Jeśli natomiast aspiryna* straciłaby działanie przeciwbólowe, to jej produkcja trwałaby nadal – jako leku rozrzedzającego krew.

Propozycja ta rozwiązuje problem *ontological stacking*, jednakże teoria ukonstytuowania nie spełnia przynajmniej jednego z omówionych wcześniej kryteriów adekwatności ontologii dla artefaktów technicznych – zasady ograniczenia realizowalności. Stwierdza ona, że ontologia ta powinna umożliwiać rozumienie możliwości wyciągania wniosków na temat materialnej bazy z twierdzeń na temat artefaktów i ich funkcji oraz *vice versa*. Oczywiście, mając daną materialną bazę oraz sprzyjające okoliczności, można wyciągać wnioski na temat właściwej funkcji artefaktu. Cóż jednak począć z wnioskowaniem odwrotnym – od artefaktu i jego funkcji do materialnej bazy? W teorii ukonstytuowania twierdzenia dotyczące funkcji artefaktu pociągają za sobą jedynie to, że przestrzennie zbieżny agregat jest „właściwy", to znaczy zawiera obiekty o „odpowiedniej" strukturze, które zostały zebrane „jak należy". Jednak określenia typu: „właściwy", „odpowiedni" czy „jak należy" niewiele mówią na przykład o tym, co takiego powoduje, że samolot może latać. Wiemy jedynie, że został on zbudowany z odpowiednich materiałów we właściwy sposób.

Teoria ukonstytuowania wymaga zatem pewnych ulepszeń i wyjaśnień, by na jej podstawie móc precyzyjnie określać związek pomiędzy artefaktem technicznym i jego materialną bazą. Można nakładać dodatkowe warunki na agregat – bądź też przyjrzeć się bliżej okolicznościom, w których struktura fizyczna tworzy obiekt, dodając do nich okoliczności, związane na przykład z agentem.

Podsumowując, można stwierdzić, że obecnie żadnej z istniejących ontologii artefaktów technicznych nie sposób po prostu „zastosować" do obiektów nazywanych komputerami. Co więcej, współcześnie coraz trudniej jest odróżnić komputery w rozumieniu „maszyny" od wykonywanych na nich programów. Istotne więc staje się jednoznaczne określenie, czym jest *hardware* (sprzęt), a czym *software* (oprogramowanie)[10].

1.1.2. Hardware *versus* software

Rozróżnienie pomiędzy powyższymi pojęciami związane jest z wieloma szczególnie trudnymi, ważnymi i dalekosiężnymi problemami[11]. Na

---

[10] Pojęcia: „hardware" i „software" używane są w języku polskim w wersji oryginalnej lub też tłumaczone są odpowiednio jako sprzęt i oprogramowanie, co z filozoficznego punktu widzenia może być mylące. Pozostajemy zatem przy wersji angielskiej tych terminów.

[11] Rozróżnienie hardware/software ma na przykład swoje konsekwencje prawne, ponieważ w większości państw (nie tylko europejskich) oprogramowanie jest chronione prawem autorskim, a urządzenia prawem patentowym.

pierwszy rzut oka sprawa jest prosta: hardware to maszyna, a software – zbiór instrukcji, który czyni maszynę operatywną w określony sposób. Zastanawiając się głębiej nad powyższym zagadnieniem, nie sposób jednak nie sformułować szeregu istotnych pytań. Sprecyzujmy je.

Czy omówione rozróżnienie dotyczy tylko komputerów, czy też wszystkich maszyn? Czy komputerami nazwiemy wszystkie urządzenia działające zgodnie z pewnym zbiorem instrukcji? Kwestia przekręcenia pokrętła w budziku: czy jest to już „zaprogramowanie" budzika, a on sam jest rodzajem komputera? I dalej – czym są instrukcje dla maszyny? Bo jeśli są one jedynie pomyślane lub napisane ołówkiem na kartce, to nie mogą – jeszcze – wpływać na ich bezpośrednie zachowanie. Ale jeśli są już dane w postaci materialnej (płyty CD, karty pamięci) i stały się tym samym częściami maszyn (zostały do nich wprowadzane), to dlaczego nie określimy ich pojęciem „hardware"?

W literaturze spotkać można różne próby definiowania pojęcia „hardware" – zazwyczaj uznawane jest ono za dopełniające do pojęcia „software". Eden i Turner (2006) w swej taksonomii obiektów informatyki podają następującą definicję:

> *Hardware*$_{\text{DEF2}}$ kategoria maszyn liczących, których zachowanie może być modelowane przez uniwersalną maszynę Turinga i których zbiór instrukcji jest zakodowany w jakimś języku pełnym w sensie Turinga.

Oczywiście tak zdefiniowany obiekt kategorii *Hardware* może być wbudowany do obiektu, który do tej kategorii nie należy. A zatem mikroprocesor wbudowany w mikrofalówkę to obiekt kategorii *Hardware*, ale sama mikrofalówka do tej kategorii nie należy. Co więcej, obiekty kategorii *Hardware* są materialne, podczas gdy obiekty kategorii *Programy*[12] (czyli software) takie nie są. Mikroprocesory są obiektami fizycznymi, które mają określoną masę i zajmują miejsce w czasie i przestrzeni, a programy nie posiadają żadnej z tych własności.

Czy rozróżnienie to jest rzeczywiście tak proste i sprowadza się do fizycznych cech sprzętu, których programy nie posiadają? Niestety, nie.

Po pierwsze, postęp technologiczny powoduje, że to, co kiedyś uważane było za software, obecnie określa się jako „hardware" – pewne operacje wcześniej obsługiwane programistycznie obecnie realizowane są sprzętowo. Na przykład wiele skryptów wczesnych języków programowania (języków maszynowych) zawierało podprogram (software) dodawania dwóch liczb zmiennoprzecinkowych; pojawiły się jednak mikroprocesory (hardware) wykonujące operacje zmiennoprzecinkowe, i podprogramy stały się

[12] Definicje kategorii *Programy* znaleźć można w paragrafie 2.1.1.

zbędne. Zatem software może być zastąpiony przez hardware jedynie w taki sposób, że są one funkcyjnie nierozróżnialne.

Po drugie, istnieją programy (tak zwane *firmware*), które wbudowywane bezpośrednio w urządzenia stają się ich nieodłączną częścią – zapewniają podstawowe procedury ich obsługi. W komputerze programem takim jest BIOS (*Basic Input/Output System*) – zapisany w pamięci stałej komputera (w pamięci ROM płyty głównej lub na kartach rozszerzeń, na przykład na kartach graficznych), pośredniczący pomiędzy sprzętem a systemem operacyjnym. Programy firmware są zatem przykładami „hardware, który jest software". Jak zatem odróżnić, który jest który?

Definicja relacji równości obiektów, nazywana w literaturze prawem Leibniza, stwierdza, że *x* jest tożsame z *y* zawsze i tylko wtedy, gdy dla wszystkich własności *F*, *x* ma własność *F* zawsze i tylko wtedy, gdy *y* ma tę własność. Określenie to zawiera w sobie dwie zasady: zasadę identyczności nierozróżnialnych (jeśli dla wszystkich własności *F*, *x* ma własność *F* zawsze i tylko wtedy, gdy *y* ma własność *F*, to *x* i *y* są identyczne) oraz zasadę nieodróżnialności identycznych (jeśli *x* jest identyczny z *y*, to dla wszystkich własności *F*, *x* ma własność *F* zawsze i tylko wtedy, gdy *y* ma własność *F*). Są one powszechnie uważne za kluczowe dla określenia równości obiektów. Zgodnie z tym prawem do stwierdzenia odmienności dwóch obiektów wystarczy wskazanie cech, które posiada jeden z nich, a których nie posiada drugi. Pozwala to na odróżnienie software od hardware, gdyż nawet jeśli procesor wykonuje te same obliczenia co program, to jednak posiada on cechy (takie jak masa czy też podatność na korozję), których ten drugi nie posiada.

Różnic pomiędzy software a hardware można również szukać głębiej – poza cechami fizycznymi. Słowo „software" sugeruje, że w programach jest coś „miękkiego" (ang. *soft* oznacza miękki, delikatny), czego nie ma w sprzęcie (hardware, ang. *hard* oznacza twardy). Co oznacza owa „miękkość"?

Suber (1998) twierdzi, że hardware jest programowalny, a software przenośny. Rozumie on przy tym pojęcie przenośności szeroko, zarówno jako możliwość wykonania programu na różnych maszynach, jak i jako możliwość przenoszenia programów z jednego nośnika na inny: z papieru na dysk magnetyczny, z dysku do pamięci RAM komputera i tak dalej. Cecha ta oddaje zatem niezależność programów od ich nośników oraz od sprzętu. Tak więc firmware to nie software, ponieważ nie jest przenośny.

Jak już wspomniano, najczęściej definiuje się hardware w opozycji do software, twierdząc, że software jest pojęciem abstrakcyjnym, a hardware – obiektem konkretnym. I tak, charakteryzuje się hardware jako obiekt fizyczny stanowiący część systemu komputerowego[13], co sugeruje, że software

[13] Przez system komputerowy rozumiemy połączenie sprzętu i programów na nim wykonywanych.

nie jest częścią takiego systemu lub – co gorsza – że nie jest on „fizyczny" (należy bowiem pamiętać, że program komputerowy można rozważać zarówno jako obiekt abstrakcyjny, jak i na poziomie fizycznym[14]).

Sprowadzanie powyższego do poziomu abstrakcyjne/konkretne było wielokrotnie krytykowane. James Moore (1978) uważa, że dychotomia software/hardware jest czysto pragmatyczna. Dla jednej osoby dany fragment systemu stanowi „software" (ponieważ osoba ta może go zmienić), a dla innej „hardware". Pisze on: „Dla danej osoby i systemu komputerowego software to programy, które mogą być uruchomione w tym systemie oraz – zawierające instrukcje, które osoba ta może zmienić; hardware natomiast to pozostała część systemu" (s. 215).

Zatem dla specjalisty piszącego programy w języku maszynowym hardware to na przykład obwody systemu, ale już dla programisty używającego języka wysokiego poziomu pojęcie hardware rozszerza się i zawierają się w nim również programy w języku maszynowym. Większość użytkowników natomiast prawie cały system traktuje jako hardware.

Przytoczone wątpliwości Subera i Moore'a dotyczące rozróżnialności software/hardware krytykuje Duncan (2009). Twierdzi on, że aby sprostać powyższemu wyzwaniu, konieczne jest przyjęcie szerszej perspektywy ontologicznej.

W celu dokonania ontologicznego rozróżnienia software/hardware Duncan proponuje przyjęcie Basic Formal Ontology (BFO)[15], która z założenia jest ontologią realistyczną (jej terminy reprezentują rzeczywistość) oraz uniwersalną (niezależną od dziedziny, jaką opisuje). Podstawowym terminem tej ontologii jest byt rozumiany jako to, co istnieje w czasie i w przestrzeni. Byty można podzielić na dwie kategorie: ciągłe (ang. *continuant*) oraz pojawiające się (ang. *occurrent*). Duncan (2009) opisuje je następująco:

> 1) ciągły: byt, istniejący w całości w pewnym czasie, w którym istnieje zupełnie, utrzymuje się przez długi czas, zachowując swoją tożsamość, i nie ma elementów czasowych.
> Przykłady: serce, osoba, kolor pomidora, masa chmury
>
> 2) pojawiający się: byt, który ma elementy czasowe i który staje się, ujawnia lub rozwija w czasie. […]
> Przykłady: życie organizmu, najciekawsza część życia Van Gogha, czasoprzestrzenny obszar zajmowany przez guz nowotworowy (s. 17).

---

[14] Kwestię tę omówiono szerzej w paragrafie 2.1.1.

[15] The Basic Formal Ontology (BFO) jest ontologią zaprojektowaną dla integracji różnych dziedzin naukowych. Nie zawiera zatem terminów fizycznych, chemicznych, biologicznych i innych, które mogłyby nie pasować do innych dziedzin, a które to terminy obejmują ontologie nauk szczegółowych. Pełny opis BFO można znaleźć na stronie http://www.ifomis.org/bfo.

Przez „hardware" autor rozumie byt fizyczny wchodzący w skład systemu komputerowego i stanowiący jednolitą całość. Proponuje on nazywanie takich bytów „egzemplarzami sprzętu komputerowego" (ang. *piece of computing hardware*; Duncan, 2009, s. 18). Z kolei „software" to byt zaprojektowany w celu umieszczenia go w wielu systemach komputerowych, znajdujący się w jakimś obiekcie fizycznym (płyta CD, karta pamięci i tym podobne). Autor określa go mianem „oprogramowania" (ang. *software application*). Tak rozumiane software i hardware należą do kategorii bytów ciągłych, ponieważ zachowują swoją tożsamość w czasie.

BFO dzieli byty ciągłe na dwa rodzaje:

1) niezależne – nośniki jakości, którym przysługują lub w których tkwią inne byty, a które same w sobie nie przysługują niczemu,
2) zależne – byty, które są albo zależne od jakiegoś bytu niezależnego, albo przysługują bytowi niezależnemu.

Przykładem bytu niezależnego, podanym przez autora, są włosy, natomiast ich kolor jest już bytem zależnym, ponieważ przysługuje bytowi niezależnemu. A więc, zgodnie z takim ujęciem, wszystkim instancjom bytów niezależnych przysługują byty zależne – i na odwrót.

Dalej, byty zależne zostały podzielone na dwa rodzaje:

1) rodzajowo zależne (ang. *generically dependent*) – zależne od jakiegoś nośnika, a nośniki te mogą się zmieniać,
2) specjalnie zależne (ang. *specifically dependent*) – zależne zawsze od tego samego nośnika.

Plik w formacie pdf jest przykładem bytu rodzajowo zależnego, ponieważ może istnieć na wielu komputerach i wiele razy w czasie. Czerwony kolor pomidora natomiast jest przykładem bytu specjalnie zależnego, gdyż jeśli nie istnieje konkretny pomidor, to nie istnieje również ta jego cecha, którą jest czerwony kolor.

Przyjęcie powyższej klasyfikacji bytów, zaczerpniętej z BFO, pozwala na jednoznaczne odróżnienie software od hardware. Hardware jest bytem ciągłym, niezależnym, realizującym funkcje konieczne do wykonywania obliczeń. Na przykład dysk twardy służy do przechowywania danych na nośniku magnetycznym, a pewien zbiór bramek logicznych w procesorze – do dodawania liczb całkowitych. W obu wypadkach istnieje pewna funkcja obliczeniowa (przechowywanie informacji i dodawanie liczb) przysługująca tym obiektom.

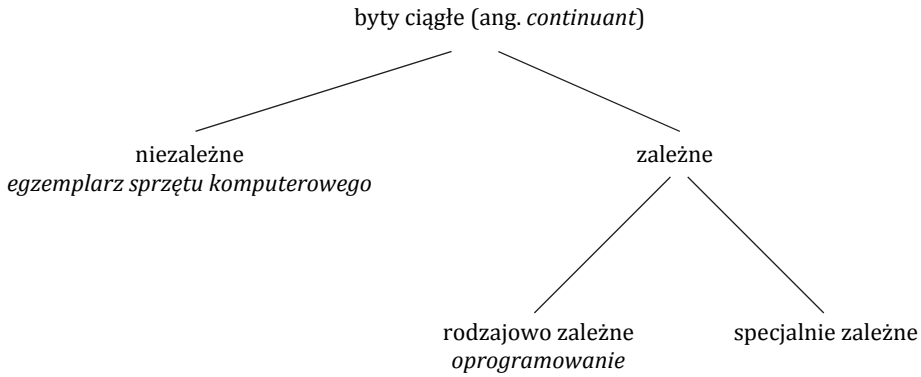
Zastanówmy się teraz, gdzie w przedstawionej powyżej hierarchii bytów jest miejsce dla software. Program jest zbiorem zakodowanych instrukcji reprezentujących pewną funkcję obliczeniową. Instrukcje takie kodowane są w różny sposób, ponieważ software istnieje w wielu formach: na płytach CD, kartach pamięci i tak dalej. Ponadto zakodowany zbiór instrukcji nie jest funkcją obliczeniową, lecz jej reprezentacją. Ta sama funkcja może być

zapisana w różnych językach programowania, a więc za pomocą różnych kodów. Zatem software jest przykładem bytu rodzajowo zależnego.

Duncan (2009) podaje następujące definicje hardware i software:

> 1) egzemplarz sprzętu komputerowego: niezależny byt ciągły, który jest celowo zaprojektowany dla aktualizacji jednej lub więcej funkcji obliczeniowych.
> […]
> 3) oprogramowanie: rodzajowo zależny byt ciągły, kodujący reprezentację jednej lub więcej funkcji obliczeniowych (funkcji realizowalnych) (s. 24).

Definicje te wskazują miejsce omawianych bytów w hierarchii BFO, które można przedstawić schematycznie w następujący sposób:



Rys. 1. Klasyfikacja bytów ciągłych według BFO

Z powyższego schematu wynika, że wbrew opisanym wcześniej sugestiom Moore'a i Subera, software i hardware są różnymi, jednoznacznie zaklasyfikowanymi bytami.

Problemem, który pojawia się zarówno przy pojęciu hardware, jak i software, jest określenie właściwych metod sprawdzania ich poprawności. Przejdźmy zatem do omówienia sposobów weryfikacji działania urządzeń, jakimi są komputery.

1.1.3. Weryfikacja komputerów jako urządzeń

Współczesne systemy komputerowe składają się z urządzeń nazywanych komputerami oraz z całej gamy programów – od systemów operacyjnych aż po specjalistyczne aplikacje inżynieryjne, księgowe czy gra-

ficzne[16]. Komputery te są bardzo wiarygodne i rzadko się psują, jednakże trudno jest sprawdzić ich wiarygodność i poprawność działania. Wiąże się to w dużej mierze z poziomem złożoności nowoczesnych systemów. Ponadto testowanie, którym się je poddaje, może jedynie wykazać w danym systemie obecność błędów, a nie ich brak (Dijkstra, 1989). Jedyną metodą udowodnienia stuprocentowej poprawności działania systemu byłaby jego weryfikacja, czyli podanie odpowiedniego dowodu formalnego. Jednak stworzenie takiego dowodu wydaje się niemożliwe lub przynajmniej krańcowo trudne. Tak więc „[…] połączenie komputer+program zaskakuje nas zarówno tym, co robi, jak i tym, czego nie robi [….]" (Bornat, 2006, s. 2).

Prace nad formalną weryfikacją urządzeń rozpoczęły się niemal równocześnie z badaniami związanymi z weryfikacją programów. Obecnie szczególnie ważne jest sprawdzanie sprzętu używanego w aplikacjach związanych z szeroko pojmowanym bezpieczeństwem, takich jak: kontrola lotów, systemy kontroli i ograniczania dostępu (na przykład w bankach), bądź też jak medyczne systemy diagnostyczne. Można także wskazać przyczyny natury ekonomicznej, dla których warto weryfikować urządzenia przed ich wytworzeniem – na przykład w przypadku źle skonstruowanych chipów przebudowywanie ich jest procesem długim i kosztownym (Cohn, 1989). Między innymi z powyższych powodów w latach 80. XX wieku National Security Agency finansowała badania „*provable secure*", w rezultacie których bezpieczeństwo komputerów byłoby zagwarantowane matematycznie. Chodziło o stworzenie procedur formalnego dowodzenia tego, że w systemie bezpieczeństwa tkwiącym w projekcie nie ma luk i błędów. Jednym z efektów tych prac było powstanie wojskowego mikroprocesora o nazwie VIPER (***V****erifiable* ***I****ntegrated* ***P****rocesor for* ***E****nhanced* ***R****eliability*).

VIPER był mikroprocesorem zaprojektowanym przez W. Cullyera, C. Pygotta oraz J. Kershawa w placówce naukowo-badawczej ministerstwa obrony Wielkiej Brytanii. Dla projektantów od początku było oczywiste, że VIPER zostanie formalnie zweryfikowany. Ich wysiłki zaowocowały stworzeniem specyfikacji wyrażonej w języku formalnym oraz bardzo prostym projektem (na przykład „przerwania" były zabronione, a lista instrukcji została ograniczona do minimum). Sam proces weryfikacji odbywał się na Uniwersytecie w Cambridge w latach 1986-1987.

W materiałach promocyjnych nazywano VIPER-a pierwszym dostępnym komercyjnie mikroprocesorem z formalną specyfikacją i dowodem na to, że chip jest z nią zgodny. Analizując te materiały, można odnieść mylne

---

[16] Programy komputerowe pojawiają się również w wielu urządzeniach, które trudno nazwać komputerami, na przykład w samochodach, robotach przemysłowych, telefonach komórkowych i tym podobnych.

wrażenie, że udowodnienie bezpieczeństwa projektu pociąga za sobą twierdzenie, iż wszystkie urządzenia zbudowane zgodnie z nim będą bezpieczne.

Avra Cohn (1989), jeden z członków zespołu VIPER-a, stwierdza, że istnieją liczne ograniczenia związane z formalną weryfikacją sprzętu. Po pierwsze, chip jako urządzenie fizyczne nie może zostać formalnie sprawdzony. Sprawdzeniu takiemu podlega jedynie jego projekt. Nie można zatem powiedzieć, że chip zachowuje się w zamierzony sposób. Cohn pisze: „Ani zamierzone zachowanie, ani fizyczny chip nie są obiektami, do których można zastosować słowo 'dowód' w sensowny sposób. Zarówno intencja, jak i chip mogą zostać nieodpowiednio opisane w języku formalnym, a to nie jest sprawdzalne" (s. 9).

Po drugie, może zdarzyć się sytuacja, że błędy w modelu abstrakcyjnym nie spowodują błędów w fizycznym urządzeniu, ponieważ osoby dokonujące weryfikacji dysponują innym opisem sprzętu niż jego wykonawcy. Być może błąd, który pojawia się w specyfikacji, nigdy nie powstał w umyśle projektantów. Tego typu sytuacje są możliwe, gdyż projektanci, wykonawcy i osoby sprawdzające sprzęt posługują się różnymi językami. Na przykład specyfikacja VIPER-a została stworzona jako hierarchia coraz bardziej abstrakcyjnych modeli wyrażonych w różnych językach. Poziom najniższy, najbardziej konkretny, został zapisany przez projektantów w języku naturalnym, w postaci tekstu wzbogaconego ilustracjami. Oczywiście przejście na poziom wyższy wymagało „przetłumaczenia" piktogramów na język formalny – a nie ma żadnej gwarancji, że tłumaczenie to było adekwatne. Nie można również sprawdzić, czy wszystkie wymagania opisane w języku naturalnym zostały właściwie oddane w formalnym języku specyfikacji. Co więcej, wszelkie wysiłki z konieczności związane z weryfikacją urządzenia ograniczają się do tych jego czynności, które zostały opisane na najbardziej abstrakcyjnym poziomie. Nie ma jednak możliwości formalnego opisu wszystkich aspektów pracy komputera.

Tym, co można udowodnić w sposób formalny, jest jedynie poprawność matematycznego modelu programu lub sprzętu, a nie właściwe działanie fizycznego urządzenia lub programu wykonywanego na fizycznej maszynie. Argumentacja matematyczna nie może bowiem pokazać odpowiedniości pomiędzy modelem matematycznym a rzeczywistością fizyczną[17]. W procesie weryfikacji urządzeń mówi się o stanie *abstrakcyjnych* miejsc w pamięci i o zachowaniu elementów wykonawczych, a nie o *rzeczywistym* urządzeniu jako takim. Błędem jest zatem twierdzenie, że dowody sprzętowe stwierdzają poprawność działania fizycznych maszyn ponad wszelką wątpliwość.

[17] Do zagadnienia tego powracamy w paragrafie 2.2.1 poświęconym sprawdzaniu poprawności programów komputerowych.

Cohn (1989) stwierdza: „Urządzenie materialne może być tylko obserwowane i mierzone; nie może być zweryfikowane. Może też ono być opisane w sposób formalny, a opis ten zweryfikowany, ale nie ma sposobu na zapewnienie trafności tego opisu" (s. 131).

Istnieje zasadnicza różnica pomiędzy naturą obiektów abstrakcyjnych a naturą obiektów fizycznych – i nie można wniosków dotyczących jednych przenosić bezpośrednio na drugie. Formalny dowód poprawności projektu urządzenia nie gwarantuje właściwego działania powstałej na podstawie tego projektu rzeczywistej maszyny.

Jeśli nawet założymy możliwość formalnego udowodnienia właściwego działania skomplikowanego systemu programów i równie skomplikowanego projektu sprzętu, to nadal nie powstanie z tego matematyczny dowód na fizyczne zachowanie się systemu komputerowego. Każdy taki „dowód" pokazuje bowiem tylko pewną relatywną poprawność, zakładając *implicite* adekwatność modeli matematycznych użytych do jego zaprojektowania.

Co więcej, istnieje wiele pozalogicznych czynników wpływających na działanie urządzeń – jak na przykład sposób i warunki ich produkcji oraz cechy otoczenia fizycznego, w którym są one używane[18]. A zatem nie jest właściwe nazywanie systemów komputerowych „zweryfikowanymi" i twierdzenie, że zawsze będą one działały poprawnie, to znaczy zgodnie ze stawianymi im wymaganiami.

Skutecznym i prostym sposobem na zwiększenie wiarygodności systemów jest ich testowanie. Obiekty fizyczne powinny być sprawdzane empirycznie poprzez przeprowadzenie odpowiednich testów, obserwację wyników oraz ich interpretację (Turner, 2011). Jeśli któryś z testów nie powiedzie się choćby raz, to wówczas można stwierdzić, że system nie działa prawidłowo. Z kolei jeśli dane wejściowe zostały wybrane odpowiednio – przyjmuje się, że urządzenie fizyczne spełnia swoją specyfikację. Czasami, choć bardzo rzadko, możliwe jest przetestowanie wszystkich danych wejściowych. Jednak nawet wtedy stwierdzenie poprawności działania systemu jest jedynie stwierdzeniem empirycznym, dotyczącym obiektu fizycznego, a nie twierdzeniem matematycznym. Poprawność artefaktów fizycznych to zawsze kwestia empiryczna, a ta nie daje absolutnej pewności, że urządzenie będzie działało.

Debata nad dopuszczalnymi metodami weryfikacji systemów komputerowych wywołała dyskusję dotyczącą statusu dowodów matematycznych. Naukowcy szukają odpowiedzi na pytania, co to jest dowód matematyczny i jakie są jego cechy, oraz czy formalne weryfikacje urządzeń i programów

[18] Tym samym wnioskowania dotyczące poprawnego działania urządzeń są ze swej natury podważalne, podobnie jak rozumowania dotyczące weryfikowalności programów.

można uznać za dowody matematyczne. Niektórzy z nich twierdzą, że problem związany z definiowaniem dowodu matematycznego jest nie tylko zagadnieniem filozoficznym czy metodologicznym, lecz także problemem całkowicie realnym. Ujawnił się on wraz z wprowadzeniem na rynek wspomnianego już VIPER-a. Czy stwierdzenie reklamy, że istnieje dowód poprawności jego działania, jest prawdziwe? Czy inni producenci sprzętu mogą domagać się sprostowania tej informacji – na przykład, na drodze sądowej? MacKenzie (1992) twierdzi, że wątpliwości dotyczące cech dowodu matematycznego mogą zostać w niedługiej przyszłości rozstrzygnięte właśnie w sądzie. Już teraz znaleźć można pewne rozwiązania tej kwestii, i to nie w pracach naukowych dotyczących filozofii, matematyki czy informatyki, ale w dokumentach rządowych. I tak, „UK Interium Defence Standard 00-55" rozstrzyga, co jest dowodem formalnym, a co ścisłą argumentacją:

> Dowód formalny to ściśle określony ciąg formuł logicznych, w którym każda z nich wynika z formuły wcześniejszej w tym ciągu – lub też jest przykładem aksjomatów teorii logicznej.
> Ścisła argumentacja pojawia się w literaturze naukowej na poziomie argumentacji matematycznej, gdy zostanie ona poddana recenzji (ocenie środowiska) (...)[19].

Rozumienie komputera jako urządzenia fizycznego ma jednak niewiele wspólnego z takimi pojęciami, jak dowód formalny czy ścisła argumentacja.

Inaczej rzecz ma się w przypadku traktowania komputerów jako obiektów abstrakcyjnych – maszyn matematycznych.

1.2. Komputer jako obiekt abstrakcyjny

Teoretyczne pojęcie „komputer" stworzyli logicy matematyczni w celu ułatwienia badań nad naturą obliczalności, a zwłaszcza nad jej ograniczeniami (Mahoney, 2002). Można zatem uważać, że komputery są maszynami matematycznymi, a każdy aspekt ich funkcjonowania da się wydedukować ze ścisłej definicji – i to z matematyczną precyzją, za pomocą praw czystej logiki (Hoare, 1986).

Jedną z takich ścisłych matematycznych definicji podał Alan Turing (1936), a abstrakcyjny byt przez niego opisany nazywa się dzisiaj maszyną Turinga. Składa się on z nieskończonej taśmy złożonej z klatek, w których mogą być zapisywane symbole, oraz z przesuwającej się po niej głowicy

[19] UK Ministry of Defence Directorate of Standarization, Glasgow 5 April 1991 (UK Interium Defence Standard 00-55), w: (McKenzie, 1992).

wykonującej instrukcje (stanowiące program maszyny). W każdym momencie maszyna znajduje się w jednym z możliwych stanów, głowica natomiast „obserwuje" jedną z klatek taśmy. Działanie maszyny jest bardzo proste. Głowica czyta symbol znajdujący się na taśmie i – zgodnie z instrukcjami programu – pozostawia go bez zmian bądź też zamienia na inny, jednocześnie przesuwając się na niej o jedną klatkę (w lewo lub w prawo). Następnie maszyna może przejść w inny stan. Wykonując kolejne instrukcje programu, byt ten „oblicza".

Oczywiście, formalizm maszyny Turinga nie jest jedyną próbą doprecyzowania pojęcia „komputer" jako obiektu abstrakcyjnego. Istnieje wiele innych jego formalnych (matematycznych) opisów. Wszystkie one posiadają jedną wspólną cechę: definiują go jako obiekt wykonujący operacje na symbolach, a więc w terminach czysto syntaktycznych. „Komputery operują na symbolach" – to zdanie zupełnie oczywiste dla informatyka, kontrowersyjne jednak z punktu widzenia filozofa. Zdefiniowanie komputerów jako urządzeń wykonujących operacje na symbolach wiąże się bowiem z różnymi trudnościami.

Charakterystyka komputerów w terminach czysto syntaktycznych, „w terminach ustalenia (ang. *assignment*) zer i jedynek" (Searle, 1990, s. 5), jest w pewnych sytuacjach niewystarczająca. Pomijana w takich opisach semantyka jest istotna – na przykład w definiowaniu identyczności systemów, koncentrującym się nie na ich budowie, lecz na rodzaju obliczeń, jakie wykonują[20].

Co więcej, taki sposób rozumienia „bycia komputerem" jest zbyt ograniczony. Żaden z istniejących opisów formalnych nie obejmie bowiem wszystkich komputerów. Przedstawiona powyżej maszyna Turinga oddaje strukturę bardzo niewielu współczesnych maszyn liczących. Można oczywiście rozszerzać istniejące – lub tworzyć nowe – opisy abstrakcyjne, ale „proces ten nigdy w sposób właściwy nie uchwyci pojęcia bycia komputerem, tak jak żaden katalog z biżuterią nie zdefiniuje pojęcia 'biżuteria'" (Hayes, 1997, s. 3).

Z drugiej strony, czysto syntaktyczny opis jest za szeroki, gdyż pozwala na objęcie tą nazwą zbyt wielu systemów fizycznych. I tu stajemy wobec problemu wskazanego przez Searle'a (1990), określanego jako „uniwersalna realizowalność": wszystko może być komputerem, ponieważ dla każdego obiektu można znaleźć taki jego opis, zgodnie z którym obiekt ten jest cyfrowym komputerem. Każdy system fizyczny można bowiem rozumieć jako realizację jakichś abstrakcyjnych bytów formalnych, takich jak zbiory czy równania różniczkowe. Nie powinniśmy jednak traktować każdego takiego systemu jako komputera.

---

[20] Por. semantyczna koncepcja obliczania (Shagrir, 1999).

Przyjęcie, że istotne cechy komputera są czysto syntaktyczne, prowadzi do możliwości różnych jego realizacji. Na przykład fizycznymi realizacjami maszyny Turinga mogą być zarówno mechaniczny kalkulator zbudowany z kół zębatych, system hydrauliczny, przez który przepływa woda, jak i cała gama współczesnych komputerów zbudowanych z wykorzystaniem różnych technologii[21]. Jeśli ponadto obliczanie definiuje się w terminach ustalenia (przypisania) składni, to wszystko może być cyfrowym komputerem, ponieważ o każdym obiekcie posiadającym wystarczającą liczbę stanów można myśleć jako o wykonującym obliczenia, odpowiednio tylko je interpretując. Zatem zdaniem Searle'a, twierdzenie, że coś jest komputerem lub że pracuje obliczeniowo – jest po prostu bezsensowne.

Dla każdego programu komputerowego istnieje pewien dostatecznie skomplikowany obiekt, dla którego istnieje opis, zgodnie z którym obiekt ów jest implementacją tego programu. Na przykład ścianę można interpretować jako realizację edytora tekstu Wordstar, ponieważ istnieje pewien wzorzec ruchów molekuł, izomorficzny z formalną strukturą tego programu[22]. Searle (1990) pisze: „Dla każdego obiektu istnieje taki jego opis, zgodnie z którym obiekt ten jest cyfrowym komputerem" (s. 5).

Warto zauważyć, że składnia nie jest nieodłącznie związana z fizyką, a przypisanie składni do obiektu jest zawsze relatywne w stosunku do obserwatora, traktującego pewne zjawiska fizyczne jako syntaktyczne. Nie można zatem obiektywnie stwierdzić, że coś jest komputerem, albowiem charakterystyka obiektu jako komputera cyfrowego jest zawsze subiektywna.

Przedstawione powyżej trudności związane z definiowaniem komputera jako maszyny abstrakcyjnej mogą sugerować, że „bycie komputerem" jest pierwotne w stosunku do pojęcia maszyny Turinga lub innej maszyny abstrakcyjnej. Trudności te mogą także motywować do poszukiwań innej charakterystyki komputera.

1.3. Inne koncepcje komputera

Jedną z podstawowych motywacji do poszukiwań charakterystyki komputera, która unika pojęcia maszyny abstrakcyjnej, jest chęć wyeliminowania problemu uniwersalnej realizowalności – czyli wykluczenia poza

---

[21] Weizenbaum (1976) podaje przepis na skonstruowanie komputera z rolki papieru toaletowego i stosu małych kamyczków.

[22] Searle twierdzi nawet, że jeśli ściana jest dostatecznie duża, to może ona implementować dowolny program, nawet ten zapisany w ludzkim mózgu.

zakres pojęcia „komputer" takich obiektów, jak ściana, kamień czy system trawienny, które w potocznym rozumieniu komputerami nie są.

George McKee (w: Hayes, 1997) proponuje nazywanie komputerem czegoś, co może być programowalne. Pojęcie programowalności rozumie on przy tym w sposób bardzo szeroki, jako możliwość wpływania informacji zewnętrznych na zachowanie urządzenia. To oczywiście eliminuje z zakresu tego pojęcia wspomnianą wcześniej ścianę, nie do końca jednak rozwiązuje problem. Jest to bowiem jedynie przesunięcie akcentu z pojmowania komputera jako maszyny obliczającej na traktowanie go jako maszyny programowalnej, samego pojęcia programowalności natomiast autor nie wyjaśnia. Blisko z nią związana jest również idea rozumienia komputerów jako systemów wykonujących algorytmy, to znaczy realizujących ich formalne specyfikacje (Copeland, 1996).

Naukowcy definiujący pojęcie „komputer", pragnący uniknąć problemu uniwersalnej realizowalności, koncentrują się na doprecyzowaniu pojęcia obliczalności. Powszechne jest bowiem rozumienie komputera jako urządzenia wykonującego obliczenia. I tak, Stufflebeam (w: Hayes, 1997) nazywa nim tylko to, co implementuje pewną funkcję. Jeśli jednak rozumieć funkcję jako obliczenie wykonywane na określonych danych w celu otrzymania wyniku – tak jak dzieje się to w przypadku teorii obliczeń – to wiele komputerów tego nie robi. Pełnią one bowiem zadania kontrolne – chociażby nadzorując procesy wykonywane w czasie rzeczywistym.

A zatem termin „obliczanie" należy rozumieć szerzej, mówi się bowiem o obliczeniach zarówno na maszynach abstrakcyjnych (maszyna Turinga „oblicza"), jak i na konkretnych urządzeniach fizycznych nazywanych komputerami. Hayes (1997) pisze: „Przez 'komputer' rozumiem maszynę, która przeprowadza obliczenia lub która oblicza. W tym rozumieniu maszyna Turinga nie jest komputerem, ale matematyczną abstrakcją pewnej jego odmiany" (s. 2).

Niezależnie od tego, czy pojmujemy komputer jako urządzenie fizyczne, czy też jako obiekt abstrakcyjny, to związek pojęcia „komputer" z pojęciem „obliczalność" jest zawsze bardzo bliski. Co więcej, ponieważ termin „komputer" odnosi się do całej gamy różnych obiektów, warto poszukać definicji „obliczania" pasującej do nich wszystkich.

Można próbować, jak to czyni Berkeley (w: Hayes, 1997), określić zakres pojęcia „obliczanie", kierując się kryterium poziomu abstrakcyjności. Podaje on definicję bardzo abstrakcyjną: „ObliczenieT (T – techniczne). Obiekt Toblicza, jeśli wykonuje zgodne z regułami manipulacje na złożonych ustrukturyzowanych symbolach" (s. 9), a z drugiej strony definicję blisko związaną z konkretem, jakim jest komputer: „ObliczanieC (C-common

sense). Coś jest Cobliczalne, jeśli jest to coś, co może być wykonane przez komputer" (s. 9).

Definicja Cobliczania ma dwie zasadnicze wady. Po pierwsze, brak w niej doprecyzowania pojęcia „komputer". Przyjęcie, że jest to urządzenie wykonujące Cobliczenia, prowadzi do tak zwanego błędnego koła (komputer wykonuje Cobliczenia, a Cobliczenia to te, które mogą być przez niego wykonane). Po drugie, jeśli przyjmiemy taką definicję, to zbyt wiele obiektów moglibyśmy uznać za Cobliczalne. Przykład: komputery mogą wykonać (zagrać) dowolny utwór muzyczny – przyjmijmy, że będzie to V Symfonia c-moll Beethovena, ale czy tylko z tego powodu powinniśmy nazywać tę symfonię Cobliczalną? Propozycja Berkeleya nie jest wolna od problemu uniwersalnej realizowalności, lecz przenosi go na inny poziom: z mówienia o komputerach wykonujących obliczenia – do mówienia o samych tych obliczeniach.

Przedstawione powyżej opisy działania komputera, koncentrujące się (podobnie jak w przypadku maszyny Turinga) na obliczaniu, określa się w ten sposób jako „z punktu widzenia procesora" (jednostki liczącej). Patric Hayes (1997) skupił się na innym istotnym elemencie komputera – na jego pamięci.

Pamięć ta zawiera wzorce (również wzorzec pusty) ulegające zmianom. Hayes porównuje ją do „magicznego papieru", na którym zapis może się samoistnie zmieniać – lub na którym mogą pojawiać się nowe napisy. Różne rodzaje komputerów definiuje się wówczas abstrakcyjnie w terminach składni języka ich wzorców, rodzaju zmian syntaktycznych, których mogą dokonywać, oraz teorii wiążącej wzorce z tymi zmianami.

Definiowanie takie jest, zdaniem Hayesa, odpowiednie z kilku powodów. Po pierwsze, podkreśla, że komputery wykonują operacje na wzorcach w zależności od ich formy, jednak bez dodatkowego założenia, że czynności te nie odnoszą się w jakiś sposób do denotacji symboli. Po drugie, pozwala ono na abstrahowanie od szczegółów technicznych, związanych z działaniem maszyny, ponieważ jest to opis na wyższym poziomie. Po trzecie, wskazuje, że możliwość wykonywania operacji (symbolicznych) na wzorcach jest istotną cechą każdego komputera. Tym, co powoduje, że maszynę nazywamy komputerem, jest między innymi fakt, iż wykonuje ona operacje symboliczne. Po czwarte, podkreśla, że pojęcie „bycia komputerem" obejmuje więcej niż tylko odpowiednie przechodzenie pomiędzy stanami. Wielu maszyn zmieniających stany nie nazywamy komputerami, ponieważ nie kodują one symboli i nie wykonują na nich operacji.

Hayes twierdzi zatem, że komputer jest maszyną operującą na wzorcach „wejściowych" i zamieniającą je zgodnie z instrukcjami na inne, „wynikowe". Jak jednak opisać komputer, który nigdy nie został uruchomiony?

Pojęcie „zmiany wzorca" zaproponowane przez Hayesa nie jest w tym wypadku odpowiednie, ponieważ nie ma żadnego wzorca „wejściowego" podlegającego przekształceniu. Istvan Berkeley (w: Hayes, 1997) dla uniknięcia powyższej trudności proponuje stwierdzenie, że *X* jest komputerem w przypadku, gdy *możliwe jest*, by *X* wykonał obliczenia. Konieczny jest wówczas właściwy wybór modalności. Część naukowców twierdzi, że możliwość logiczna jest zbyt słaba, i proponuje możliwość fizyczną (por. np. Bringsjord, w: Hayes, 1997), inni natomiast (np. Hartcastle, w: Hayes, 1997) opowiadają się za formą pośrednią pomiędzy możliwością fizyczną a ludzką. Jak się wydaje, żadna z zaproponowanych dotychczas definicji nie obejmuje przypadku komputera, „ który nigdy nie został włączony".

Może zatem, zgodnie z powszechną intuicją, należy ograniczyć się do definiowania „działającego komputera" jako urządzenia fizycznego?

1.4. Podsumowanie

W rozdziale tym omówiono pojęcie komputera nierozerwalnie związanego z informatyką. Przedstawione tu analizy pokazują, że jest ono trudne do zdefiniowania ze względu na swoją wieloznaczność. Próby odpowiedzi na nawet najprostsze pytania wiążą się z wieloma trudnościami.

Wydaje się, że powszechnie wiadomo, czym jest fizyczne urządzenie nazywane komputerem i kto jest jego wynalazcą. Przecież wszyscy stykamy się na co dzień z komputerami, rozumianymi jako samodzielne maszyny lub też jako części innych urządzeń (samochodów, telefonów, sprzętu gospodarstwa domowego i tym podobnych). W podrozdziale 1.1 przedstawiono trudności związane ze zdefiniowaniem komputera jako urządzenia fizycznego, rozpoczynając od ujęcia historycznego. Rozumienie terminu „komputer" zmieniało się bowiem w czasie.

Za pierwsze komputery można uznać abaki (deski rachunkowe) i liczydła, wykorzystywane przez starożytnych rachmistrzów do przeprowadzania obliczeń; trudno wtedy wskazać ich wynalazcę. Również „zegar liczący", zbudowany przez Wilhelma Schickarda w XVII wieku, pretenduje do miana pierwszego komputera, ponieważ był on pierwszym mechanicznym urządzeniem liczącym (służącym do dodawania i odejmowania oraz ułatwiającym mnożenie i dzielenie). Takie ujęcie związane jest z rozumieniem komputera jako urządzenia wykonującego obliczenia; nie jest ono powszechne.

Obecnie mianem komputera określa się maszyny uniwersalne, zaprojektowane do wykonywania szerokiej klasy zadań, nieograniczających się tylko do prostych obliczeń. Autorem koncepcji maszyny uniwersalnej jest Charles

Babbage, który od roku 1833 rozpoczął prace nad programowalną „maszyną analityczną". Maszyna ta, nazywana „dziadkiem współczesnych komputerów", była napędzana mechanicznie. Jeśli zatem uważać komputer za urządzenie elektroniczne, to jego wynalazcą nie będzie Babbage.

Wskazanie twórcy pierwszego tak właśnie rozumianego komputera nie jest łatwe. Ze względu na czas powstawania tych urządzeń – lata II wojny światowej i tuż po jej zakończeniu – oraz związane z nimi utajnienie danych, trudno obecnie stwierdzić z całą pewnością, czy wynalazcą komputera jest Konrad Zuse, czy może twórcy brytyjskich maszyn Colossus.

Warto w tym miejscu przypomnieć, że przez wiele lat za pierwszy komputer uważano amerykańskiego ENIAC-a, którego konstruktorzy próbowali nawet uzyskać patent na „komputer". Jednak sąd w Minneapolis wniosek odrzucił, a za twórcę pierwszego komputera uznał Johna Vincenta Atanasoffa, który w latach 30. XX wieku zbudował maszynę o nazwie ABC. Pretensji do autorstwa wynalazku komputera nie zgłaszał natomiast John von Neumann, który wniósł istotny wkład w jego powstanie, publikując w 1945 roku opis koncepcji programu wprowadzanego. Stała się ona podstawą tak zwanej architektury von Neumanna, a ta z kolei – teoretyczną podstawą budowy większości współczesnych komputerów. Czym jednak jest współczesny „komputer"?

Inżynierowie pod pojęciem „komputer" rozumieją najczęściej maszynę ogólnego przeznaczenia, do przetwarzania informacji lub symboli. Nie ma jednak powszechnej zgody co do podstawowych jego własności, na przykład co do tego, czy musi to być urządzenie cyfrowe. Trudno jest zatem określić cechy komputera jako urządzenia fizycznego, a tym bardziej stworzyć jego ontologię.

Można szukać ontologii komputerów – wytworów człowieka służących do realizacji określonych celów – wśród teorii artefaktów. Z punktu widzenia filozofii informatyki najciekawszą kategorią artefaktów są artefakty techniczne, ponieważ zalicza się do nich komputery. Próbom określenia ontologii komputerów jako artefaktów technicznych oraz związanym z tym trudnościom poświęcono paragraf 1.1.1.

Artefakty techniczne wydają się mieć naturę dualną, są bowiem obiektami fizycznymi (cechy fizyczne decydują o realizowanych przez nie funkcjach) oraz intencjonalnymi zarazem (funkcje te odnoszą się do czegoś niematerialnego – ludzkich intencji). Dotychczas nie powstała żadna powszechnie akceptowana teoria tak rozumianych artefaktów, która łączyłaby oba ich aspekty: fizyczny i intencjonalny (funkcjonalny). Najbardziej problematyczne wydaje się być zdefiniowanie kluczowego pojęcia funkcji artefaktu. W paragrafie 1.1.1 omówiono podstawowe koncepcje funkcji artefaktów technicznych: przyczynową, intencjonalną oraz etiologiczną.

Przeprowadzone analizy pokazują, że żadnej z istniejących obecnie ontologii artefaktów technicznych nie można bez zastrzeżeń, to znaczy bez koniecznych uzupełnień i wyjaśnień, zastosować w przypadku urządzeń nazywanych komputerami.

Z pojęciem komputera związana jest ciekawa z filozoficznego puntu widzenia kwestia, która ma również swoje konsekwencje praktyczne (por. przypis 11), a mianowicie problem rozróżnienia hardware/software. W paragrafie 1.1.2 omówiono różne propozycje jego rozwiązania.

Eden i Turner (2006) w swojej taksonomii obiektów informatyki definiują hardware jako kategorię maszyn liczących, których zachowanie może być modelowane przez uniwersalną maszynę Turinga oraz których zbiór instrukcji jest zakodowany w jakimś języku pełnym w sensie Turinga. Podkreślają przy tym, że obiekty kategorii *Hardware*, w przeciwieństwie do obiektów kategorii *Programy*, są materialne. Rozróżnienie to, chociaż wydaje się zgodne z powszechnie akceptowanymi intuicjami, okazuje się niewystarczające. Sprowadzanie rozróżnienia hardware/software do dychotomii fizyczne/abstrakcyjne jest obecnie często krytykowane. Aby go uniknąć. Duncan (2009) proponuje przyjęcie uniwersalnej ontologii realistycznej nazywanej BFO, która pozwala zdefiniować software i hardware jako różne, jednoznacznie zaklasyfikowane byty.

Paragraf 1.1.3 poświęcony został metodom sprawdzania poprawnego działania komputerów. Współczesne systemy komputerowe, rozumiane jako połączenie sprzętu i oprogramowania, są bardzo złożone. Najczęściej poddawane są one testowaniu, czyli empirycznemu sprawdzeniu dla reprezentatywnego zbioru danych. Jednak przeprowadzenie nawet najbardziej rozbudowanych testów nie daje stuprocentowej pewności co do poprawności działania systemu, ponieważ może jedynie wykazać obecność w nim błędów, a nie ich brak.

Można spotkać się z twierdzeniem, że stworzenie formalnego dowodu poprawności systemu (czyli weryfikacja) zapewnia jego niezawodne działanie. Trudno się jednak zgodzić z tym poglądem. Tworzenie dowodów formalnych jest bowiem procesem długotrwałym i bardzo trudnym, a jego rezultat nie gwarantuje właściwego działania urządzenia w praktyce, gdyż sprawdzeniu formalnemu można poddać jedynie jego projekt (matematyczny model), a nie sposób działania zbudowanej w oparciu o ten projekt fizycznej maszyny. Co więcej, matematyczny dowód pokazuje tylko poprawność modelu, a nie jego adekwatność, to znaczy zgodność z rzeczywistością fizyczną. Warto pamiętać również, że istnieje wiele zewnętrznych czynników wpływających na działanie urządzeń (na przykład warunki ich produkcji i eksploatacji), których nie można nawet przewidzieć, a co dopiero poddać weryfikacji. Wydaje się zatem, że sprawdzenie prawidłowego

działania urządzenia fizycznego, jakim jest komputer, ma niewiele wspólnego z dowodzeniem formalnym.

Istnieją jednak inne wyjaśnienia terminu „komputer", które nie ograniczają się jedynie do fizycznych maszyn. W podrozdziale 1.2 omówione zostały próby zdefiniowania go za pomocą środków matematycznych, można bowiem uważać, że komputery są maszynami abstrakcyjnymi, których zachowanie da się wydedukować z ich ścisłej definicji. Istnieje wiele formalnych (matematycznych) opisów komputera, w tym pojęcie maszyny Turinga, jednak wszystkie definiują go jako obiekt wykonujący operacje na symbolach, a więc w terminach czysto syntaktycznych. Takiej ujęcie budzi wiele kontrowersji wśród filozofów.

Pomijana w matematycznych definicjach komputera semantyka jest w wielu przypadkach bardzo istotna, na przykład w definiowaniu identyczności systemów, koncentrującym się na rodzaju wykonywanych przez nie obliczeń. Ponadto żaden z istniejących opisów formalnych nie obejmuje wszystkich obecnie produkowanych komputerów. A co więcej, z czysto syntaktycznymi ich opisami związany jest sformułowany przez Searle'a (1990) problem uniwersalnej realizowalności, rozumiany jako stwierdzenie, że wszystko może być komputerem, ponieważ dla każdego obiektu możliwe jest znalezienie takiego jego opisu, zgodnie z którym jest on realizacją jakichś abstrakcyjnych bytów formalnych (na przykład zbiorów). Wszystkie trudności związane z definiowaniem komputera jako maszyny abstrakcyjnej wykonującej obliczenia, w szczególności problem uniwersalnej realizowalności, są motywacją dla badaczy do poszukiwań innej charakterystyki komputera.

W podrozdziale 1.3 omówiono kilka innych koncepcji komputera. Zamiast definiować komputer jako maszynę wykonująca obliczenia, można twierdzić, że jest ona programowalna (McKee, 1997) lub też że wykonuje algorytmy (Copeland, 1996). Takie ujęcia nie rozwiązują jednak problemów, lecz przenoszą je na inny poziom, z mówienia o problematycznym pojęciu „obliczania" na definiowanie komputera w równie niejasnych terminach „programowalności" i „algorytmu". Można również szukać innego rozwiązania dla problemu uniwersalnej realizowalności – nie poprzez wyeliminowanie pojęcia „obliczalności", lecz przez doprecyzowanie go. Ciekawa wydaje się również propozycja Hayesa (1997), który definiując komputer, skupia się nie na obliczeniach, lecz na zmianach w jego pamięci, którą porównuje do „magicznego papieru", na którym mogą pojawiać się nowe napisy, a istniejące mogą ulegać zmianie.

Przedstawione w tym rozdziale próby określenia istoty komputera, pomimo że wiążą się z wieloma trudnościami, nie są bezsensowne. Istnieją bowiem rzeczywiste urządzenia nazywane komputerami i trudno jest za-

przeczyć twierdzeniu, że nie wszystko jest komputerem. Niezależnie od tego, czy zgodzimy się z poglądem, że komputery są obiektem badań informatyki, czy też nie, to próby ich zdefiniowania – również jako obiektów abstrakcyjnych – prowadzą do głębszego rozumienia natury związanych z nimi pojęć, na przykład obliczania, programu czy algorytmu. Te wszechobecne urządzenia stawiają zatem przed filozofami nowe, liczne wyzwania, choć – jak twierdzą niektórzy – nazwać informatykę „nauką o komputerach" to tak, jakby nazwać chirurgię „nauką o nożach".

Rozdział 2

Program komputerowy

Jednym z głównych zadań informatyków jest tworzenie programów. To one sprawiają, że komputery jako urządzenia są użyteczne, kierują bowiem ich działaniem i powodują, że komputery „robią" to, czego oczekujemy. Można zatem stwierdzić, że programy pisane są w odpowiedzi na zapotrzebowanie użytkowników komputerów oraz że służą rozwiązywaniu problemów w szerokim rozumieniu tego słowa.

Truizmem będzie stwierdzić, że komputery bez programów są praktycznie bezużyteczne. Różnorodność dostępnego oprogramowania powoduje, że współczesne komputery są uniwersalne[1], zaś systemy komputerowe, rozumiane jako połączenie sprzętu i oprogramowania, tak powszechnie używane i wykorzystywane w wielu różnych dziedzinach życia człowieka.

Każdy informatyk potrafi pisać programy, umiejętność ta jest bowiem nieodłączną częścią wszystkich programów kształcenia w tym zawodzie[2]. Niektórzy twierdzą nawet, że tworzenie oprogramowania jest cechą definiującą informatykę jako dyscyplinę, ponieważ informatycy zajmują się teorią i praktyką programowania komputerów. Khalil i Levy (1978) piszą: „[b]udowanie programów, opisywanie programów, katalogowanie własności języków programowania, obserwowanie zachowania programów: to dokładnie to, czym jest aktywność nazywana 'informatyką' […]" (Bornat, 2006, s. 4).

Programowanie nie jest prostą czynnością, lecz złożonym procesem, który rozpoczyna się od opisania celów, dla jakich tworzony jest program.

---

[1] Por. z definicją komputera jako maszyny uniwersalnej przedstawioną w podrozdziale 1.1.

[2] Współcześnie nie jest to jednak domena wyłącznie informatyków. Wielu specjalistów innych dziedzin, np. biologów, chemików, inżynierów, pisze programy odpowiadające ich potrzebom najczęściej związanym z symulacjami komputerowymi.

Opis taki – nazywany specyfikacją – może być wyrażony zarówno w języku naturalnym, jak i w odpowiednim formalizmie. Następnie tworzy się metodę rozwiązania danego problemu lub też wybiera się wśród istniejących algorytmów, po czym następuje jego implementacja, czyli zapis w odpowiednim języku programowania.

Komputer – a właściwie jego procesor – operuje wyłącznie na zerach i jedynkach oraz „rozumie" polecenia zapisane w tak zwanym języku maszynowym. Jednak współcześnie programiści nie muszą znać języków procesorów, ponieważ mają do dyspozycji całą gamę języków programowania wysokiego poziomu – one to całkowicie zwalniają twórców z konieczności myślenia o programie na poziomie procesora i jednocześnie pozwalają na używanie takich abstrakcyjnych pojęć, jak zmienna czy procedura. Zatem większość programów pisana jest obecnie z użyciem języków programowania wysokiego poziomu, a powstający w ten sposób kod źródłowy – tj. zapis programu – jest dla człowieka bardziej czytelny i łatwiejszy do zrozumienia niż kod maszynowy.

Jednakże kod źródłowy, niezależnie od tego, w jakim języku programowania został napisany, należy „przetłumaczyć" na postać binarną (tj. kod maszynowy). Takiego tłumaczenia można dokonać na dwa sposoby. Pierwszym jest kompilacja, czyli automatyczne tłumaczenie kodu źródłowego na kod maszynowy za pomocą specjalnych programów nazywanych kompilatorami. Tak powstały kod binarny można wykonywać wiele razy bez konieczności ponownego używania kompilatorów – kompilacja jest bowiem czynnością jednorazową. Inaczej dzieje się w przypadku drugiego sposobu tłumaczenia kodów źródłowych, który polega na wykorzystaniu interpreterów. Programy te na bieżąco (linia po linii) odczytują kod, analizują go i wykonują jego kolejne, przetłumaczone już fragmenty. W tym przypadku każdorazowe wykonanie programu związane jest z koniecznością użycia interpretera.

Kolejnym bardzo ważnym etapem procesu tworzenia oprogramowania jest sprawdzenie jego poprawności. Można tego dokonać na wiele sposobów, poczynając od stworzenia – jeśli to możliwe – dowodu poprawności programu zbliżonego formułą do dowodu matematycznego, a kończąc na testowaniu, czyli eksperymentalnym sprawdzeniu działania programu dla reprezentatywnego zbioru danych wejściowych. Zadanie to jest jednym z najszerzej dyskutowanych problemów filozoficznych i praktycznych związanych z informatyką[3]. Powrócimy do niego w paragrafie 2.2.1.

---

[3] Pomijamy tu pozostałe etapy tworzenia oprogramowania – między innymi wdrożenie i utrzymanie aplikacji – gdyż ich analiza niczego nie wnosi do rozważań filozoficznych związanych z informatyką.

W rozdziale tym przeanalizujemy zagadnienia filozoficzne związane z opisanym powyżej procesem tworzenia oprogramowania, a także przedstawimy wybrane problemy ontologiczne i epistemologiczne dotyczące wytworów tego procesu – programów komputerowych. Rozpocznijmy od definicji tych ostatnich.

W trakcie przeglądania szkolnych podręczników informatyki znajdujemy zazwyczaj określenia programu komputerowego zbliżone do definicji podanej przez Grażynę Kobę (2009): „Program komputerowy to ułożony w odpowiedniej, logicznie powiązanej kolejności zestaw instrukcji dla procesora mówiących, co ma robić z dostarczonymi informacjami i w jakiej kolejności" (s. 9).

Definicja ta traktuje program jako ciąg poleceń (instrukcji) dla komputera (procesora), określających jego zachowanie podczas wykonania programu. Jednak nie każdy taki ciąg skłonni jesteśmy uznać za program komputerowy – bowiem instrukcje tworzące go powinny mieć odpowiednią formę. Mordechai Ben-Ari (2006) precyzuje pojęcie programu, określając go jako „sekwencję symboli opisującą obliczenia zgodnie z pewnymi regułami zwanymi językiem programowania". W tym przypadku w sposób naturalny pojawia się pytanie o to, czym właściwie jest język programowania i jaki jest jego związek z programem. Do kwestii tych powrócimy w dalszej części rozdziału.

Często można spotkać się z poglądem, że programy są po prostu zbiorem instrukcji tworzonych w celu rozwiązania problemu z użyciem komputera. Podkreśla się w ten sposób czysto pragmatyczne ujęcie programowania, twierdzi się, że jego efekty są narzędziami lub środowiskiem pracy ludzi. Projektuje się je w procesie uczenia się i komunikacji, by mogły jak najlepiej odpowiadać na potrzeby użytkowników (Floyd, 1987).

Wydaje się jednak, że takie rozumienie programu komputerowego jest zbyt ograniczone, ponieważ nie można charakteryzować wszystkich aktywności komputerów za pomocą terminologii rozwiązywania problemów. Istnieją bowiem obiekty nieuznawane powszechnie za programy, które jednak podpadają pod taką definicję. Zapis algorytmu, na przykład w postaci schematu blokowego, może zawierać instrukcje stworzone dla rozwiązania problemu przez komputer, jednak nie wszystkie takie schematy są programami komputerowymi. Co jest zatem cechą wyróżniającą programy spośród wszystkich ciągów instrukcji?

Moore (1978) podaje następujące określenie: „Program komputerowy jest zbiorem instrukcji, które komputer może wykonać (lub co najmniej istnieje efektywna procedura przekształcenia ich do postaci, którą może on wykonać), by zrealizować zadanie" (s. 214).

W definicji tej autor kładzie szczególny nacisk na możliwość wykonania ciągu instrukcji przez maszynę, jaką jest komputer. Akcentuje tym samym oczywisty, bliski związek programu ze sprzętem, na którym jest on wykonywany. Definicja ta uzależnia zatem pojęcie programu od pojęcia komputera, albowiem to, co uważamy za instrukcje dla niego, i w konsekwencji to, co uważamy za program, jest wyznaczone przez operacje, które może wykonać maszyna. Schemat blokowy lub inaczej – zapisany ciąg instrukcji – możemy zatem traktować jako program tylko wtedy, gdy został on przedstawiony w formie umożliwiającej jego wykonanie przez komputer. Dlatego też niektóre schematy blokowe są programami komputerowymi, a inne – nie (pozostając jedynie algorytmami). Wielką zaletą definicji Moore'a jest to, że pozwala ona na proste odróżnienie tych podstawowych pojęć informatyki: algorytmu i programu. Niestety, ma również swoje wady.

Wydaje się, że pojęcie „komputer" występujące w definicji Moore'a powinno się ograniczyć do istniejących obecnie maszyn po to, byśmy mogli uniknąć stwierdzenia, że każdy ciąg instrukcji jest programem. Istnieje przecież możliwość, że kiedyś powstanie urządzenie odpowiednie do jego wykonania. Pojęcie programu jest wówczas wyraźnie uzależnione od komputerów funkcjonujących w danym momencie – tym samym status obiektu jako programu może ulegać zmianie. Pewne ciągi instrukcji dzisiaj nie są programami (gdyż nie ma komputerów, na których można je wykonać), ale mogą stać się nimi w przyszłości. Nasuwa się więc pytanie, czy stare programy wykonywane uprzednio na nieistniejących już typach komputerów są nadal programami, czy już tylko algorytmami. W jaki sposób na rozumienie programów komputerowych wpływa fakt istnienia tak zwanych emulatorów – pozwalających na uruchomienie starych programów na współczesnych urządzeniach? Ten zmieniający się w czasie status obiektów jako programów komputerowych przeczy powszechnej intuicji związanej z tym pojęciem. Każdy informatyk wydaje się rozumieć w praktyce, co to jest program komputerowy i w jaki sposób wpływa on na działanie komputera. Jednak niewielu informatyków zdaje sobie sprawę, jak ciekawym jest on bytem z punktu widzenia filozofa.

2.1. Ontologia

Tradycyjnie w filozofii wyróżnia się dwa podstawowe rodzaje bytów: abstrakcyjne i konkretne. Zazwyczaj nie mamy problemów z zaklasyfikowaniem większości obiektów spośród tych, z jakimi mamy do czynienia na co dzień. Urządzenie, jakim jest komputer, podobnie jak samochód i drzewo, jest przykładem bytu konkretnego (fizycznego). Z kolei liczby, zbiory

i algorytmy są obiektami abstrakcyjnymi. Gdzie w tym podziale znajduje się miejsce dla programów komputerowych? Czy są one może obiektami o szczególnym statusie, które nie pasują do tego klasycznego podziału?

2.1.1. Dualna natura programów komputerowych

Aby odpowiedzieć na pytania dotyczące statusu ontologicznego programu komputerowego, powinniśmy najpierw wyraźnie odróżnić program jako ciąg instrukcji zapisanych w języku programowania od tego, który jest nieodłącznie związany z maszyną fizyczną. Na rozróżnienie to po raz pierwszy zwrócił uwagę James Fetzer (1989) i dlatego nazywa się je często „dwuznacznością Fetzera".

Program rozumiany jako ciągi instrukcji jest bytem abstrakcyjnym, nieistniejącym w czasie i przestrzeni, a charakteryzowanym tylko za pomocą relacji logicznych. Może on zostać zapisany na różnych nośnikach fizycznych: od kartki papieru, poprzez karty i taśmy dziurkowane (używane w dawnych komputerach), dyski magnetyczne, optyczne i inne. Nośniki takie są oczywiście obiektami fizycznymi, stąd w literaturze pojawia się czasem stwierdzenie, że program komputerowy może być serią kart dziurkowanych lub taśmą magnetyczną, lub jedną z wielu innych form (por. np. Moore, 1978). Wydaje się, że twierdzenia te wynikają z nieodróżniania samego programu (bytu abstrakcyjnego) od jego nośnika (sposobu fizycznego zapisu).

Należy również odróżnić zapis programu w języku programowania (kod źródłowy) od jego wykonania na konkretnych urządzeniach. Znakomicie pojęli to Ammon H. Eden i Raymond Turner (2006), którzy stworzyli taksonomię obiektów informatyki. Jednym z celów przyjętych przez autorów było stworzenie takiego opisu obiektów informatyki (w tym programów), który oddawać będzie ich rozumienie zarówno przez informatyków, jak i przez laików.

Informatycy mają do czynienia z trzema kategoriami obiektów: *Hardwarem*, *Metaprogramami* oraz *Programami.* Obiekty tej ostatniej dzielą się na dwie grupy, odpowiadające wspomnianemu powyżej rozróżnieniu. Są to:

- *Skrypty* – byty, które zawierają poprawnie sformułowane instrukcje dla pewnej klasy komputerów cyfrowych;
- *Boty* – obiekty powstające przez uruchomienie skryptu w konkretnych warunkach fizycznych (nazywane w systemach operacyjnych „wątkami" – ang. *threads*), które są, w odróżnieniu od skryptów, bytami czasowymi.

Programy rozumiane jako napisy, nazywane skryptami, definiuje się w następujący sposób: „*Skrypt* $_{\text{DEF4}}$ Kategoria bytów S_L („S_L jest programem")

takich, że *L* jest językiem programowania pełnym w sensie Turinga oraz *S* jest dobrze sformułowanym wyrażeniem w *L*" (Eden i Turner, 2006, s. 4).

Takie określenie odnosi pojęcie skryptu do pojęcia języka programowania, z podkreśleniem, że język ten nie może być dowolny, lecz powinien być pełny w sensie Turinga (ang. *Turing-complete*), to znaczy musi zawierać wszystkie konstrukcje potrzebne do zasymulowania uniwersalnej maszyny Turinga[4]. Dzięki temu możemy wykluczyć z kategorii *Programy* pewne przypadki skrajne. Jeśli bowiem przez język programowania rozumiemy zbiór dobrze sformułowanych instrukcji dla komputera, a każdy obiekt fizyczny można opisać jako komputer w pewnym trywialnym sensie, to na przykład włącznik światła można rozumieć jako komputer, którego „język programowania" składa się z dwóch instrukcji {ON, OFF}. Warunek pełności w sensie Turinga gwarantuje w szczególności to, że język programowania obsługuje nietrywialny zbiór instrukcji[5].

Wśród skryptów wyróżnić można dwie podkategorie: *Kody Maszynowe*, czyli skrypty zapisane w języku maszynowym, i *Kody Zródlowe* – skrypty zapisane w języku programowania wysokiego poziomu. Można stwierdzić, że kody źródłowe są bardziej abstrakcyjne niż kody maszynowe w tym sensie, że skrypty napisane z języku maszynowym składają się z ciągów instrukcji interpretowanych przez komputery cyfrowe wprost, czyli bez udziału kompilatorów i interpreterów.

Jak już wspomniano, boty – byty czasowe odpowiadające pojęciu procesu lub wątku w systemie operacyjnym – są generowane poprzez uruchomienie skryptów. Kategoria *Boty* zawiera zarówno proste procesy, takie jak procesy generowane przez naciśnięcie przycisku na klawiaturze lub w mikrofalówce, jak również bardziej skomplikowane programy. Konkretny skrypt może być użyty do generowania wielu botów jednocześnie[6].

Z kategorią *Programy* ściśle związana jest kategoria *Metaprogramy*, zawierająca formuły zapisane w języku rachunku predykatów i opisujące programy. Została ona podzielona (Eden, Hirshfeld, Kazman, 2006) na trzy podkategorie: stwierdzenia strategiczne, taktyczne i implementacyjne. Nie wiadomo jednak, czy jest to pełen podział *Metaprogramów*, czy może istnieją również metaprogramy nienależące do żadnej z tych grup.

Powyższa taksonomia uwzględnia ważne z filozoficznego punktu widzenia rozróżnienie na programy jako procesy (nazywane tutaj botami)

---

[4] Pojęcie maszyny Turinga zostało omówione w podrozdziale 1.2.

[5] Powszechny jest pogląd, że wszystkie używane przez informatyków języki programowania są pełne w sensie Turinga.

[6] Współczesne systemy operacyjne pozwalają maszynom na obsługę wielu botów równocześnie.

wykonywane na konkretnych maszynach i programy rozumiane abstrakcyjnie (czyli skrypty). Pozostaje nam odpowiedzieć na pytanie o związki zachodzące pomiędzy programami-napisami a ich wykonaniami.

Jeśli traktować instrukcje programu jako zapis przyszłych stanów maszyny, na której zostanie on uruchomiony, to program komputerowy jest swego rodzaju planem czynności (aktów fizycznych), jakie komputer wykona. Można wtedy twierdzić, że program rozumiany jako proces oraz jego zapis są różnymi manifestacjami tego samego obiektu. Nawet jeśli skłonni jesteśmy uznać, że takie rozumienie związków pomiędzy programem-napisem a programem-procesem jest właściwe, to pojawić się mogą wątpliwości, kiedy przeanalizujemy inne przykłady planów i obiektów będących ich fizycznymi realizacjami. Trudno jest bowiem utożsamić plan wykładu uniwersyteckiego (na przykład zapisany na kartce) – z samym tym wykładem (wygłoszonym przed słuchaczami). Czy skłonni jesteśmy uznać je za różne manifestacje tego samego obiektu? Utożsamienie takie jest całkowicie nieintuicyjne.

Bliższe intuicji wydaje się stwierdzenie, że programy-napisy powodują lub wywołują procesy zachodzące w komputerze. Zatem obiekty tekstowe, jakimi są kody maszynowe programów, w pewien sposób wywołują procesy fizyczne. Powstaje jednak pytanie o naturę takiego związku przyczynowego. Co dokładnie oznacza stwierdzenie, że programy rozumiane jako napisy wywołują programy jako procesy?

Colburn (2000) nie zgadza się z tezą, że napisy (obiekty symboliczne) można łączyć z jakimikolwiek efektami przyczynowymi. Program komputerowy jest, jego zdaniem, konkretną abstrakcją (ang. *concrete abstraction*), posiadającą nośnik (medium) zapisu (tekst, który jest abstrakcją) i nośnik wykonania (konkretną realizację w półprzewodnikach). Tekst programu nie jest samym programem, lecz tylko zapisem algorytmu lub formalnym opisem procesu obliczeniowego. Jest zatem abstrakcją. Tym, co powoduje, że proces ten jest wykonywany przez maszynę, nie jest sam tekst programu, lecz jego fizyczna reprezentacja (program zapisany na nośnikach), a więc konkret.

Jeśli więc rozumiemy program komputerowy jako konkretną abstrakcję, stajemy wobec pytania, w jaki sposób obiekt taki może być jednocześnie abstrakcyjny i konkretny. Poszukując odpowiedzi, Colburn porównuje klasyczny problem filozoficzny rozpatrujący relację pomiędzy duszą a ciałem (ang. *mind-body problem*) do kwestii związku pomiędzy programem jako konkretem i programem jako abstrakcją. Analizuje on monizm i dualizm, traktując je jako możliwe odpowiedzi na powyższe pytanie. Można – w duchu monizmu – twierdzić, że program jest jednym bytem, a abstrakcja i konkret to tylko jego aspekty. Otwartym pozostaje jednak problem rozumienia pojęcia „aspekt". Nie jest też do końca jasne, czy lepsze byłoby uzna-

nie programów komputerowych za byty abstrakcyjne (idealistycznie), czy też za byty konkretne (materialistycznie). Wydaje się bowiem, że każdy z tych wyborów powoduje znaczne zubożenie potocznego rozumienia samego terminu „program". Można również, zgodnie z doktryną dualizmu, przyjąć, że program komputerowy jest bytem zarówno abstrakcyjnym, jak i konkretnym, co oczywiście nie rzuca nam światła na problem relacji pomiędzy konkretem a abstrakcją.

Jak widzimy, monizm i dualizm wydają się niewystarczające do wyjaśnienia dualnej natury programu komputerowego, w związku z czym Colburn proponuje przyjęcie zmodyfikowanej wersji zasady harmonii przedustawnej. Zgodnie z twierdzeniem sformułowanym przez Leibniza ciało i umysł tylko z pozoru wydają się wpływać na siebie nawzajem – a harmonia w ich działaniu pochodzi od Boga. Colburn (2000) pisze:

> Dla problemu abstrakcyjny/konkretny możemy zastąpić Boga programistą, który z jednej strony, przez przedstawienie algorytmu w tekście programu, opisuje świat mnożenia macierzy, zmiany wielkości okien lub nawet rejestrów procesora; ale który z drugiej strony, przez akt zapisania, skompilowania, asemblowania i linkowania, powoduje ciąg zmian stanów fizycznych, które pasują strukturalnie do jego abstrakcyjnego świata (s. 208).

Poszukując odpowiedzi na pytanie dotyczące związku pomiędzy zapisem programu a jego wykonaniem, można zwrócić się nie tylko ku filozofii umysłu (analizując związek ciała i duszy), ale również ku filozofii muzyki.

Wydaje się bowiem, że program komputerowy jako obiekt ma wiele wspólnego z utworem muzycznym. Po pierwsze, utwór jest utrwalany przez kompozytora na papierze w postaci zapisu nutowego – partytury. Stanowi ona swego rodzaju instrukcję dla wykonujących utwór muzyków. Jest więc podobna do kodu źródłowego programu komputerowego rozumianego jako zapis instrukcji dla komputera. Po drugie, utwory muzyczne wykonywane są na instrumentach muzycznych, a programy na komputerach. Zarówno instrumenty muzyczne (włączając w nie struny głosowe człowieka), jak i komputery są obiektami fizycznymi. Przy czym ten sam utwór może mieć wiele wykonań, podobnie jak program komputerowy może być uruchamiany wielokrotnie na różnych maszynach. Zdarzają się przy tym utwory nigdy niewykonane, podobnie jak programy – nigdy nieuruchomione. Po trzecie, wykonania utworów muzycznych, jak i wykonania programów komputerowych są bytami czasowymi (trwają w czasie). Cóż zatem mówią filozofowie muzyki na temat związków partytury utworu muzycznego z licznymi jego wykonaniami?

Anna Brożek (2011), chcąc podkreślić unikatowość i szczególny status ontologiczny obiektów, jakimi są utwory muzyczne, pisze:

> Partytura – to nie utwór, lecz jego zapis; utwór muzyczny jest sensem znaków w partyturze zawartych, podobnie jak powieść czy wiersz są sensem odpowiednich słów.
> Wykonanie utworu muzycznego nie jest samym utworem, lecz jego realizacją.
> […]
> Wszystko wskazuje na to, że utworów muzycznych nie da się utożsamić z niczym przestrzennym (s. 13).

A zatem kody źródłowe lub kody maszynowe to nie programy, lecz tylko zapisy programów (podobnie jak partytura jest zapisem utworu muzycznego). Potwierdza to wcześniejsze stwierdzenie o konieczności odróżnienia programów od nośników (na których zapisane są ich kody). Program jest – w powyższym rozumieniu – sensem symboli tworzących instrukcje wchodzące w skład jego kodu, natomiast jego wykonanie – które można nazywać na przykład botem – nie jest programem, lecz jego realizacją.

Każda partytura wyznacza pewną strukturę, którą realizują wszystkie adekwatne jej wykonania, a więc niektórzy filozofowie skłonni są utożsamiać utwory muzyczne z uniwersaliami. Korzystając z analogii program-utwór muzyczny, można postawić tezę, że programy komputerowe to uniwersalia, których przykładami są wszystkie poprawne ich wykonania. Jednakże utwory muzyczne, a tym samym programy komputerowe, nie są uniwersaliami w zwykłym sensie – i to przynajmniej z dwóch powodów. Po pierwsze, są one, w odróżnieniu od uniwersaliów, bytami czasowymi. Po drugie, niektóre teorie dotyczące uniwersaliów zakładają, że powstają one w wyniku abstrakcji z konkretnych przedmiotów – programy natomiast często nie są efektem takiej abstrakcji. Czym zatem są utwory muzyczne? Anna Brożek (2011) pisze:

> Są to, powtórzmy, przedmioty nieprzestrzenne, ale zarazem czasowe (kiedyś powstałe), i nieogólne (a więc niewyabstrahowane z partykulariów), a będące wytworami czynności komponowania, «uwieczniane» w partyturach i realizowane w wykonaniach.
> Ze względu na wymienione własności utwory muzyczne […] zasługują na miano „przedmiotów quasi-partykularnych" (s. 14).

Analogicznie można zaklasyfikować programy komputerowe. One również są bytami o wymiarze czasowym, które nie powstają w procesie abstrakcji z partykulariów, są zapisywane w postaci kodu źródłowego i realizowane przez wykonania na komputerach. Można je więc traktować jako kolejny, po utworach muzycznych, przykład przedmiotów „quasi-partykularnych". Jednak takie rozumienie natury programów komputerowych nie

jest powszechne. Dużo więcej zwolenników, szczególnie wśród informatyków wywodzących się z matematyki, znajduje pogląd, że programy komputerowe są bytami matematycznymi.

2.1.2. Programy jako obiekty matematyczne

Jedną z pierwszych dyskutowanych szeroko w literaturze kwestii filozoficznych związanych z informatyką jest jej związek z matematyką. Czy informatyka jest kolejną gałęzią matematyki, a jej obiekty są po prostu obiektami matematycznymi?

Zwolennicy traktowania informatyki jako nauki matematycznej (formalnej) twierdzą, że napisy tworzące program są wyrażeniami matematycznymi opisującymi z niespotykaną dokładnością zachowanie, zamierzone lub niezamierzone, komputerów, na których są wykonywane (Hoare, 1969). Co więcej, uważają, że programy rozumiane jako procesy (p) wykonywane na komputerze (jako boty) są również obiektami matematycznymi, ponieważ są w pełni opisywane przez wyrażenia matematyczne (programy rozumiane jako napisy s_p). Eden (2007) pisze: „Programy-napisy s_p są wyrażeniami matematycznymi. Wyrażenia matematyczne reprezentują obiekty matematyczne. Program p jest w pełni i dokładnie charakteryzowany przez program-napis s_p. A zatem, program jest obiektem matematycznym" (s. 145).

A skoro tak jest, to powstaje pytanie, jakiego rodzaju są to obiekty. Programy można traktować – na przykład – jako funkcje (obiekty czysto matematyczne) ze zbioru stanów początkowych (lub „wejść") w zbiór stanów końcowych (lub „wyjść"), co prowadzi do prostej analogii pomiędzy tymi programami a regułami wynikania w dowodach matematycznych (funkcjami ze zbioru przesłanek do zbioru wniosków). Fetzer ilustruje tę analogię następującą tabelą:

Tabela 1. Porównanie matematyki do programowania

	Matematyka	*Programowanie*
Dziedzina:	przesłanki	wejście (*input*)
Funkcja:	reguły wynikania	program
Przeciwdziedzina:	twierdzenia	wyjście (*output*)

Źródło: Fetzer, 1991, s. 200

Programy komputerowe można rozumieć również „metamatematycznie", czyli odpowiedniki twierdzeń matematycznych, jak to zaproponowali W. Scherlis i D. Scott (1983), przedstawiając następujący związek pomiędzy matematyką a programowaniem:

Tabela 2. Związek między matematyką a programowaniem

Matematyka	*Programowanie*
Problem	specyfikacja
Twierdzenie	program
Dowód	weryfikacja programu

Źródło: Scherlis, Scott, 1983, s. 207

Wielu filozofów uważa, że zdania dotyczące obiektów nauk formalnych (takich jak matematyka) są zdaniami analitycznymi w odróżnieniu od zdań syntetycznych (specyficznych dla nauk realnych). Jako zdania analityczne rozumie się wówczas te, które są prawdziwe na mocy znaczenia występujących w nich słów, istniejących w języku, na przykład „Jeżeli Asia jest niższa od Kasi, to Kasia jest wyższa od Asi". Zdania syntetyczne natomiast to te, których wartość logiczna nie może być ustalona bez odwoływania się do rzeczywistości innej niż językowa, na przykład „Basia Nowak ma zielone oczy".

Przyjrzyjmy się zatem zdaniom, z których zbudowane są programy komputerowe. Ich jednoznaczne zaklasyfikowanie jako analitycznych uważać można za ważny argument na rzecz pojmowania programów jako obiektów matematyki – albowiem zdania matematyki uważane są najczęściej za zdania analityczne[7].

Wszystkie współczesne języki programowania zawierają operację podstawienia, na przykład $A := 2 + 3$. Wyrażenia takie można interpretować na dwa sposoby: jako rozkazy lub jako zdania oznajmujące. W pierwszym przypadku jest to rozkaz dla procesora nakazujący, by zmiennej *A* przypisać sumę liczb 2 i 3. Oczywiście rozkazy nie są zdaniami w sensie logicznym, nie mogą więc być ani prawdziwe, ani fałszywe, a co za tym idzie – nie mogą być rozumiane jako analityczne czy syntetyczne.

Rozważmy zatem powyższy zapis jako zdanie oznajmujące, stwierdzające, że zmienna oznaczana przez *A* otrzymuje wartość odpowiedniej sumy *A* może być przy tym rozumiane albo jako oznaczenie bytu fizycznego, albo też – abstrakcyjnego. Jeśli *A* potraktujemy jako fizyczne miejsce w pamięci komputera, to operację podstawienia rozumiemy jako stwierdzenie: „fizyczne miejsce w pamięci *A* przyjmuje wartość fizycznego obliczenia 2 + 3". Jest ono więc swego rodzaju przypuszczeniem co do tego, co stanie się w pamięci komputera, gdy operacja podstawienia zostanie wykonana. Jednak, jak już wspomniano, operacje zapisane w językach programowania wysokiego

[7] W filozofii matematyki znaleźć można również inne poglądy na ten temat (por. Murawski, 1995)

poziomu (w tym operacja podstawienia) nie są wykonywane przez procesor wprost, lecz tłumaczone za pomocą specjalnych programów na podstawowe instrukcje maszynowe. Dopiero takie instrukcje wykonywane są przez procesor. A zatem zdanie reprezentujące operację podstawienia jest tylko przypuszczeniem co do wyniku wykonania programu i jest zdaniem syntetycznym, ponieważ prawdziwość tego zdania zależy od rzeczywistości. To, czy miejsce w pamięci nazywane zmienną *A* rzeczywiście przyjmie wartość sumy liczb 2 i 3, zależy bowiem od poprawności kompilacji lub – interpretacji instrukcji podstawienia oraz w sposób oczywisty od działania komputera jako maszyny.

Nasuwa się przypuszczenie, że przyjęcie interpretacji, iż programy komputerowe działają na maszynach abstrakcyjnych, gdy *A* traktujemy jako miejsce w pamięci takiej właśnie maszyny, pociąga za sobą wniosek, że występujące w tych programach instrukcje są zdaniami analitycznymi. Rozważmy za Timothym Colburnem (2000) to, w jaki sposób możemy stwierdzić prawdziwość zdania postaci: „Abstrakcyjne miejsce w pamięci *A* przyjmuje wartość abstrakcyjnego obliczenia 2 + 3".

Wyrażenie takie podobne jest do matematycznego założenia co do zawartości zmiennej, typu „niech a = 965", czyli do twierdzenia matematycznego rozpoczynającego się od „niech…", ale bez późniejszego „…wtedy". Takie zdanie nie jest ani analityczne, ani syntetyczne, co więcej, trudno mówić o jego prawdziwości – po prostu założenie zostało zrobione. Ponadto, ponieważ każde wyrażenie dowolnego języka programowania można rozumieć abstrakcyjnie, to i program komputerowy można traktować jako ciąg warunków w świecie abstrakcyjnym. Ale tak rozumiany program, podobnie jak ciąg matematycznych „niech", nie jest wyrażeniem matematycznym w żadnym interesującym sensie.

A więc – podsumowując, wyrażenia języków programowania składające się na program komputerowy są albo zdaniami syntetycznymi, albo też fragmentarycznymi wyrażeniami matematycznymi, których nie można traktować jako zdań analitycznych. Zatem analiza zdań wchodzących w skład programów nie potwierdza, że programy te są obiektami matematycznymi – w klasycznym sensie. A jeśli już mamy je traktować jako obiekty matematyki, to powinna to być matematyka stosowana, a nie matematyka czysta (Fetzer, 1988). Fetzer używa oznaczenia „PROGRAM" dla operacji, które mogą zostać wykonane na maszynie abstrakcyjnej, dla której nie ma fizycznych odpowiedników – oraz terminu „program" w odniesieniu do tych maszyn abstrakcyjnych, dla których takie odpowiedniki istnieją. Jego zdaniem istnieje analogia pomiędzy rozróżnieniem PROGRAM/program a rozróżnieniem matematyk a czysta/stosowana. Podkreśla on tym samym znaczenie ograniczeń narzucanych programom przez urządzenia fizyczne, na jakich

programy są wykonywane. Sugeruje także pewne podporządkowanie programów (obiektów abstrakcyjnych) komputerom (obiektom fizycznym).

Można także interpretować relację program-komputer na sposób odwrotny, twierdząc, że programy (byty abstrakcyjne, formalne) mają pierwszeństwo ontologiczne przed komputerami (bytami fizycznymi), na których się je wykonuje. Dijkstra (1989), jeden z prekursorów postrzegania programów jako bytów matematycznych, pisze: „Program to operowanie na symbolach abstrakcyjnych, które mogą zmienić się w konkretne przez zastosowanie do nich komputerów. Ostatecznie, nie jest celem programów instruowanie naszych maszyn, obecnie celem maszyn jest wykonywanie naszych programów" (s. 1401).

Kontynuację rozumienia programu jako obiektu abstrakcyjnego znaleźć można w koncepcji programu jako cyfrowego wzorca.

2.1.3. Programy jako wzorce

Ciekawą próbę zdefiniowania obiektu, jakim jest program komputerowy, podjął Peter Suber (1998). Twierdzi on, że program to po prostu cyfrowy wzorzec, rozumiany jako tablica komórek (lub miejsc) zawierających jeden z dwóch możliwych symboli.

Z definicją tą związane są pewne założenia ontologiczne nazywane przez autora zasadami. Pierwsza z nich to *zasada cyfrowa* stwierdzająca, że każdy wzorzec analogowy może zostać odtworzony we wzorcu cyfrowym z pewną dokładnością. Przyjęcie jej nie ogranicza jednak programów do wzorców cyfrowych. Jeśli dodatkowo przyjmiemy *zasadę wyczerpywania* zakładającą, że wzorce cyfrowe i analogowe wyczerpują dziedzinę wzorców, to „każdy rodzaj wzorca może posłużyć jako program" (Suber, 1998, s. 3).

Powyższa definicja programu komputerowego – jako bardzo ogólna – ma swoje wady. Po pierwsze, nie stwierdza, czy wzorzec musi mieć wymiar materialny, czy ma on być napisany na papierze, nagrany na dysku czy też wystarczy, by został wymyślony; a może w myśl ujęcia platońskiego: coś jest programem nawet już przed byciem pomyślanym? Po drugie, definicja ta nie podaje żadnego kryterium pozwalającego odróżnić programy od szumu (losowego układu zer i jedynek) i od danych (nie wiadomo, w jaki sposób odróżnić aktywną i pasywną rolę wzorców). Co więcej, jeśli przyjąć, jak sugeruje autor, *zasadę bezszumowości* głoszącą, że żaden wzorzec nie jest szumem dla wszystkich możliwych języków programowania i wszystkich możliwych maszyn – to każdy układ zer i jedynek można rozumieć jako program w myśl pewnej interpretacji (w odpowiednim języku programo-

wania i dla odpowiedniej maszyny). Jeśli zasada ta jest prawdziwa, to żaden wzorzec nie jest szumem z żadnej możliwej perspektywy.

Jednym ze sposób uniknięcia wymienionych problemów jest dodanie do definicji programu jako wzorca dwóch wymogów: (1) by wzorzec ten był czytelny dla maszyny oraz (2) by był wykonywalny. Jednak modyfikacja taka nadal nie pozwala na odróżnienie programu od szumu, gdyż wprowadzenie wymogu wykonywalności nie rozwiązuje kwestii, lecz jedynie przesuwa trudność z poziomu natury programu na poziom znaczenia słowa „wykonywalny". Ponadto wzorzec będący szumem także może zostać wykonany przez maszynę, należy jednak uwzględnić, że może zostać wykonany źle. Program bez błędów składniowych (możliwy do skompilowania), ale pełen błędów semantycznych (który nie wykonuje tego, czego chciał programista) jest w pewnym sensie szumem – jednak jest wykonywalny.

Aby odróżnić złe wykonania programu od dobrych, konieczne jest wprowadzenie pojęcia celu programisty. Programy realizują cele swoich autorów, natomiast kody losowe lub błędne tego nie czynią. I dalej, by zrozumieć program, musimy znać intencje, zamiary, cele lub wolę jego twórcy. Czy jednak wszystkie programy mają twórców? Co z wzorcami naturalnymi, niestworzonymi przez człowieka? Jeśli przyjmiemy, że istnieją „programy" naturalne wpisane w przyrodę, to wprowadzenie do rozważań o ich naturze pojęcia celu twórcy jest problematyczne. Prowadzi bowiem w szczególności do konieczności uznania istnienia ich twórcy – programisty przyrody (Boga?).

Przyjrzyjmy się teraz wymogowi czytelności wzorców. Aby wzorzec był czytelny dla maszyny, musi mieć właściwą formę materialną (wymaganie fizyczne) oraz – musi być zapisany w języku maszynowym (wymaganie gramatyczne). Rozważmy najpierw wymaganie fizyczne. Oczywiste wydaje się uznanie, że programem jest ciąg instrukcji zapisanych na kartce w języku naturalnym (na przykład po polsku) lub za pomocą formalizmu matematycznego. Jednak nośnik, jakim jest papier, nie gwarantuje czytelności programu dla maszyny. Zatem wymaganie fizyczne powinno zostać sformułowane w postaci bardziej ogólnej – jako stwierdzenie, że program *można* wyrazić w odpowiedniej formie fizycznej, nawet jeśli aktualnie formy takiej nie ma.

Suber twierdzi, że każdy wzorzec da się urzeczywistnić – chociażby dlatego, że można go narysować – toteż postuluje przyjęcie *zasady dostrzegalności*, mówiącej, że możliwe jest fizyczne urzeczywistnienie każdego wzorca. Wszystko, co ma reprezentację fizyczną (co jest narysowane), może być przeczytane lub odkodowane przez odpowiednio zaprojektowaną maszynę. A zatem każdy wzorzec jest czytelny dla jakiejś maszyny, czyli każdy wzorzec jest programem. Reasumując, dodanie do definicji programu jako

wzorca wymogu jego czytelności dla maszyny nie rozwiązuje problemu odróżnienia go od szumu, gdyż każdy wzorzec jest programem – w pewnym kontekście.

Co więcej, jeśli chcielibyśmy określać mianem programu wzorzec spełniający wymagania fizyczne i gramatyczne, ale zapisany na dysku twardym znajdującym się w szufladzie, to czytelność tego wzorca należy rozumieć bardzo szeroko. Programem jest bowiem również taki wzorzec, który jest czytelny dla maszyny, ale który w danym momencie nie jest przez nią czytany.

Kontynuując tę myśl, można więc twierdzić, jak to czyni Suber, że programami są również te procedury, które istnieją (jak dotąd) tylko w umyśle programisty. Mogą one bowiem zostać wypowiedziane w języku, przedstawione z pewną dokładnością w postaci cyfrowej (zgodnie z zasadą cyfrową) oraz – urzeczywistnione (zgodnie z zasadą dostrzegalności). Wówczas programami możemy nazywać nie tylko kody zapisane na dyskach, dyskietkach i innych nośnikach, które nie są obecnie wykonywane, ale również to, co jest niesprecyzowane – lub dopiero powstaje (istniejąc tylko jako idea), uznając, że uzyska odpowiednią formę w przyszłości. Programami są wtedy również niewykonywalne osobno części dużych programów (na przykład systemów operacyjnych), których to części oddzielnie maszyna wykonać nie może.

Jeśli jednak przyjmiemy zasadę dostrzegalności (wszystkie wzorce spełniają wymaganie fizyczne) oraz zasadę cyfrową (każdy wzorzec możemy przedstawić w postaci cyfrowej), to wszystkie wzorce spełniają założenie o czytelności i wykonywalności. Tak więc założenia 1) i 2), dodane do pierwotnej definicji programu, są zbędne. Co więcej, Suber (1998) zauważa, że:

> Konsekwencje tego faktu [że wszystkie wzorce są programami[8]] na pierwszy rzut oka są zaskakujące. Nie tylko losowy ciąg bitów jest programem, ale również 10^{20} cyfr liczby Pi, krzywe tworzące usta Mony Lisy, […] gwiazdy w Drodze Mlecznej, układ neuronów w mózgu. […] Wszechświat jest programem (s. 10).

Przejdźmy teraz do problemu rozróżnienia wzorca jako danych i wzorca jako programu. Suber twierdzi, że program jest wzorcem w pozycji kontrolującej, podczas gdy w innej pozycji będzie on daną (w innej konwencji językowej natomiast – szumem). Czym jednak jest owa „pozycja"? Jest to relacja pomiędzy wzorcem-programem a innym wzorcem, który obecnie funkcjonuje jako dana. Programy wpływają na działanie komputerów – podobnie jak pokrętła wpływają na działanie kół zębatych. Jednak w jaki sposób maszyna rozróżnia pozycje wzorców? W przypadku dwóch kół

[8] Przypis mój – I. B-K.

zębatych rozróżnienie, które z nich jest kierowane, a które kieruje, jest bardzo proste. Zależy ono od tego, któremu kołu pozwalamy działać najpierw. Podobnie można odróżnić wzorce jako programy, które uruchamiane są wcześniej (i pełnią funkcję kontrolną), od wzorców jako danych.

Przedstawiona powyżej koncepcja Subera nie jest wolna od zastrzeżeń. Po pierwsze, pojęcie wzorca cyfrowego, użytego do definiowania programów, wydaje się nie mniej problematyczne niż samo pojęcie programu. Autor nie odpowiada bowiem na pytanie, jaka jest natura wzorców. Bo jeśliby rozumieć je jako obiekty matematyczne, to zdefiniowanie programów jako wzorców jest tylko kolejną wersją twierdzenia, że programy są obiektami matematycznymi. Po drugie, liczne zasady pojawiające się w rozważaniach Subera podawane są zazwyczaj bez żadnego uzasadnienia; często są to bardzo silne założenia ontologiczne. Po trzecie, koncepcja Subera nie pozwala na jasne formułowanie kryterium równości programów.

Jeśli istotą programu jest wzorzec, to różne wzorce powinny stanowić różne programy. Co jednak może powodować rozumienie wzorca w terminach czysto syntaktycznych? Otóż jeśli we wzorcach zajdzie różnica choćby jednego bitu, wówczas wzorce te będą różnymi programami. Istnieją jednak odmienne wzorce, które rozumiemy jako te same programy. Na przykład kod programu w języku maszynowym, powstający przez kompilację kodu napisanego w języku programowania wysokiego poziomu, to wzorce różne, ale program ten sam. W sytuacji, gdy jeden program jest tłumaczeniem drugiego, możemy powiedzieć, że mają one ten sam wzorzec semantyczny, któremu odpowiadają dwa różne wzorce syntaktyczne. Suber uważa, że do identyfikacji programów potrzebne jest przejście od składni do semantyki. Mówiąc, że są one takie same, bardzo często myślimy o celu, dla którego zostały napisane. Myśląc o dwóch wytworach człowieka, zaprojektowanych do wypełniania tego samego celu, identyfikujemy dwa różne wzorce jako jeden program.

Aby móc postrzegać dwa różne wzorce jako ten sam program, bez odwoływania się do pojęć zewnętrznych – takich jak cel lub funkcja programu, Suber (1998) proponuje przyjęcie *zasady pitagorejskiej* mówiącej o tym, że „[s]kładnia budzi semantykę w ogólności" (s. 16). Istotą programu jest wówczas sam wzorzec. Jeśliby przyjąć ponadto zasady: dostrzegalności, bezszumowości i cyfrową, to dziedzina wzorców mogących służyć jako programy rozszerza się do wszystkiego, co można nazwać wzorcem, a pojęcie „program" i granice jego zasięgu stają się niejasne.

Nie tylko Suber poszukuje kryterium identyczności programów. Problem określenia, kiedy dwa programy można uznać za równe, jest ważny zarówno ze względu na punkt widzenia filozofii – jako jedno z podstawowych pytań ontologicznych – jak również ze względów praktycznych.

2.1.4. Kryteria identyczności programów

Określenie kryteriów identyczności programów komputerowych jest istotne nie tylko dla filozofii informatyki; ma ono bowiem również swoje skutki prawne, a to z powodu konieczności prawnej ochrony programów. Kiedy nowo powstały program uznać należy za identyczny z wcześniej istniejącym, to znaczy kiedy następuje naruszenie praw autorskich? Czy zmiana koloru lub kroju czcionki, użytej do zapisania kodów programów, wystarcza, by programy uznać za istotnie różne, czy też należy wyjść poza ich zapis i przyjrzeć się raczej efektom ich działania? W obu tych przypadkach konieczne jest doprecyzowanie i głębsza analiza pojęcia identyczności programów komputerowych.

Jeśli utożsamimy program z jego fizyczną reprezentacją, na przykład z jego zapisem w języku programowania, to zmuszeni będziemy uznać dwa programy o kodzie różniącym się tylko wyglądem czcionki za różne obiekty. Programy napisane w różnych językach programowania także należy zawsze traktować jako różne – nawet jeśli są zapisami tego samego algorytmu, a efekt ich wykonania jest identyczny. Co więcej, programu zapisanego w języku programowania wysokiego poziomu oraz – efektu jego kompilacji (lub interpretacji), który również jest programem, nie można uznać za identyczne, ponieważ mają one różne kody (różne zapisy), a jest to niezgodne z intuicją. Należy zatem szukać innego kryterium identyczności uwzględniającego raczej znaczenie instrukcji programu, a nie tylko sposoby jego zapisu.

Narzędzi do precyzyjnego określenia takich kryteriów dostarczyć może tak zwana semantyka języków programowania. Języki programowania, rozumiane jako zbiór abstrakcyjnych definicji i reguł[9], podobnie jak języki naturalne mają swoją gramatykę, według założeń której możemy określić, jakie wyrażenia są syntaktycznie poprawne. Zazwyczaj wykorzystuje się w tym celu gramatyki regularne, ale nie opisuje się znaczenia tych wyrażeń. Oprócz gramatyki języki mają również semantykę określającą znaczenie występujących w nich konstrukcji. I dopiero gramatyka wzbogacona o semantykę (formalną lub nieformalną) pozwala informatykom na badanie własności języków programowania oraz programów. Semantyki języków programowania służyć mogą do ich klasyfikowania oraz pomagają w tworzeniu zupełnie nowych języków. Dostarczają również narzędzi do badania

[9] Język programowania (a właściwie notację programistyczną) można rozumieć jako system formalny w sensie Fregego i Russella. Działający program można wtedy traktować jako konstruktywny dowód na to, że pewne obliczenie jest możliwe w danym systemie formalnym (Bornat, 2006).

samych programów – w tym do określania kryteriów ich identyczności oraz poprawności.

Semantyka języków programowania była początkowo rozwijana przez grupę matematyków i informatyków o zainteresowaniach filozoficznych. Programowanie pierwszych komputerów polegało na wprowadzaniu do nich wprost instrukcji zapisanych w języku maszynowym. Powstałe w latach 50. XX wieku języki programowania wysokiego poziomu[10] pozwoliły programistom na zapisywanie instrukcji dla komputerów w sposób bardziej zrozumiały i całkowicie abstrahujący od sposobu ich wykonania przez komputer. Trudności w projektowaniu tychże języków wynikały między innymi z braku jasno sprecyzowanej semantyki, bez której trudno jest mówić na przykład o tym, co uważać należy za poprawną implementację danego języka.

Pierwsze kroki ku semantyce języków programowania zrobiono w latach 60. XX wieku. Semantyki częściowo wyrażane były – i nadal są – w języku potocznym. Dla każdej konstrukcji językowej podaje się wówczas słowny opis jej znaczenia. Na przykład opis instrukcji warunkowej może wyglądać następująco: „wykonanie instrukcji warunkowej polega na zbadaniu wartości logicznej warunku, po czym – wykonaniu pierwszej z podanych instrukcji, jeśli warunek był prawdziwy, bądź też – instrukcji drugiej, jeśli warunek był fałszywy i jednocześnie fraza else była obecna" (Bylina, Bylina, 2011, s. 27). Taki sposób opisu semantyki języka programowania, choć szeroko rozpowszechniony, jest jednak bardzo nieścisły.

W bardziej ścisłych, formalnych semantykach teoretycznych, nazywanych językami specyfikacji, wykorzystuje się pewne podobieństwo języków programowania wysokiego poziomu do języków formalnych – takich jak logika pierwszego rzędu. Po pierwsze, występują w nich zmienne, do których można przypisywać wartości, predykaty oraz funkcje. Po drugie, wiele z podstawowych operacji programistycznych można rozumieć jako przypisanie wartości zmiennym. Języki programowania mają jednakże pewne cechy odróżniające je od języków logiki; na przykład występujące w nich zmienne mają typy, pozwalające na przypisywanie im tylko wartości określonego rodzaju. Z punktu widzenia semantyki tych języków nie ma jednak możliwości przedstawienia tych typów, ponieważ w elementarnej teorii mnogości zbiory są kolekcjami obiektów bez dodatkowej struktury. Co więcej, większość języków programowania pozwala programom na wykonywanie czynności, które zmieniają wartości zmiennych bądź też kończą się innymi nieodwracalnymi skutkami (mówi się wtedy, że powodują one efekty uboczne). Wszystkie te cechy niezwykle utrudniają tworzenie formalnej semantyki dla języków programowania.

---

[10] Fortran powstał w roku 1954, Algol w latach 1958-1960, a Lisp w latach 1958-1962.

Współcześnie spotkać możemy trzy podstawowe typy takich semantyk: 1) operacyjne – opisujące każdą strukturę języka przez czynności, jakie jej odpowiadają (zwykle na jakiejś maszynie abstrakcyjnej), 2) denotacyjne – w których znaczenie każdego zapisu w danym języku opisuje się za pomocą funkcji matematycznych, oraz 3) aksjomatyczne – traktujące frazy języka jako obiekty matematyczne charakteryzowane poprzez aksjomaty.

Semantyki operacyjne (Landin, 1964; Plotkin, 1981) do interpretacji składni języka programowania wykorzystują pojęcie maszyny abstrakcyjnej, tłumacząc jego wyrażenia na ciąg instrukcji dla niej. Podstawowe jest tutaj pojęcie „konfiguracji", które można rozumieć jako rzeczywisty stan maszyny realizującej programy rozważanego języka. Wśród konfiguracji wyróżnia się początkową, od której zaczyna się realizacja programu, oraz końcowe, które odpowiadają jego zakończeniu. Definiuje się też dla niej relację przejścia modelującą pojedynczy krok maszyny realizującej program. Obliczeniem jest wówczas ciąg kolejnych takich kroków, z których każdy następny rozpoczyna się w tej konfiguracji, do której doprowadził krok poprzedni. W ten sposób sprowadza się instrukcje programu do obliczeń (rozumianych jako relacje matematyczne) na maszynie abstrakcyjnej.

Również semantyki denotacyjne do interpretacji wyrażeń języka programowania korzystają z obiektów matematycznych (Milne i Strachey, 1977). Najczęściej używa się w tym celu teorii mnogości oraz teorii kategorii, tworząc dla każdej struktury składniowej denotację, to znaczy funkcję określającą wartość danego wyrażenia (definiującą jego sens). Wartości nadawane są całej gamie różnych obiektów: od zmiennych i stałych poprzez fragmenty programów (nadające wartość zmiennym) aż po cały program. Podprogram również może być traktowany jak rodzaj funkcji, której argumentami są wartości semantyczne jego parametrów, a wartością jest wartość semantyczna wyrażenia, które podprogram zwraca. Nie jest jednak wówczas możliwe odróżnienie dwóch podprogramów, które obliczają tę samą wartość – choć w inny sposób, na przykład jeden z nich pracuje na zmiennych lokalnych, a drugi powoduje zmianę wartości zmiennych globalnych. Takie fragmenty kodu nie są wzajemnie zastępowalne, a mogą mieć tę samą wartość semantyczną. Problem ten nazywa się „nieprzezroczystością referencyjną" (White, 2004).

Zarówno semantyki operacyjne, jak i denotacyjne bazują na obiektach matematycznych, takich jak zbiory czy funkcje. Powstaje zatem pytanie, czy istnieje między nimi istotna różnica koncepcyjna. Turner (2007) odpowiada kategorycznie – nie, powołując się na fakt, że zarówno semantyki operacyjne, jak i denotacyjne wprowadzają interpretacje matematyczne. Maszyna abstrakcyjna jest bowiem obiektem matematycznym określonym w terminach teoriomnogościowych. Potrzebna jest zatem dokładniejsza

analiza tych dwóch rodzajów semantyk, która pozwoli określić ich wzajemne relacje.

Powróćmy teraz do problemu określenia równości programów komputerowych w kontekście ich wartości semantycznych. Można twierdzić (Turner, Eden, 2011), że każda semantyka języka programowania wyznacza kryterium równości napisanych w tymże języku programów, ponieważ dwa programy są równe, jeśli mają tę samą wartość semantyczną, to znaczy:

$$P = Q \text{ wtedy i tylko wtedy, gdy } ||P|| = ||Q||,$$

gdzie $||P||$ oznacza wartość semantyczną programu P.

Przyjrzyjmy się następującym programom obliczającym funkcję silnia:

Program 1	Program 2

function Silnia (*n*: **integer**): **integer begin**
  if *n* = 0
  then Silnia:= 1;
  else
  Silnia := (*n*) * Silnia(*n*-1);
end;
	function Silnia (n: **integer**): **integer var**
	  x, *y*: **integer**;
	begin
	  y := 1;
	  x := 0;
	  while *x*<*n* **do begin**
	  x := *x*+1;
	  y := *y***x*;
	  end;
	  Silnia := *y*;
	end;

Określenie równości tych dwóch programów zależy od przyjętej semantyki. Pewna semantyka denotacyjna może je utożsamiać, ponieważ obliczają one tę samą funkcję – silnię, całkowicie abstrahując od sposobu obliczania jej wartości. Z kolei semantyka operacyjna może rozróżniać programy 1 i 2, jeśli weźmiemy pod uwagę poszczególne kroki obliczeń. Na szczęście istnieje jednak zasada pozwalająca rozstrzygnąć, która semantyka jest odpowiednia dla określenia równości programów.

Powszechnie przyjmowanym kryterium służącym do rozróżniania obiektów jest prawo Leibniza[11], które zawiera w sobie dwie zasady: zasadę identyczności nierozróżnialnych (jeśli dla wszystkich własności *F*, *x* ma własność *F* zawsze i tylko wtedy, gdy *y* ma własność *F*, to *x* i *y* są identyczne) oraz zasadę nieodróżnialności identycznych (jeśli *x* jest identyczny z *y*, to

[11] Prawo to zostało omówione w paragrafie 1.1.2.

dla wszystkich własności *F*, *x* ma własność *F* zawsze i tylko wtedy, gdy *y* ma własność *F*). Ich odzwierciedlenie znajdujemy w pojęciu obserwacyjnej równoważności programów – z filozoficznego punktu widzenia rozumianej jako rodzaj funkcjonalizmu.

Definicja brzmi: dwa programy P i Q są *obserwacyjnie równoważne* wtedy i tylko wtedy, gdy we wszystkich przypadkach, gdy wejścia programów P i Q są takie same, to również ich wyjścia są takie same.

Inaczej: programy P i Q są obserwacyjnie równoważne wtedy i tylko wtedy, gdy we wszystkich kontekstach C[•], w których C[P] jest poprawnym programem, również C[Q] jest poprawnym programem o tej samej wartości semantycznej.

I tak na przykład dwa programy wykonujące operacje na bazach danych są obserwacyjnie równoważne, w odniesieniu do tej samej semantyki operacji bazodanowych, wtedy i tylko wtedy, gdy wykonanie ich w tych samych kontekstach (na maszynach o tej samej architekturze, w tym samym systemie operacyjnym i tak dalej) prowadzi do takich samych efektów w bazie danych.

Oczywiście, w praktyce nie jest możliwe sprawdzenie zachowania programu we wszystkich możliwych kontekstach – dlatego też do pojęcia obserwacyjnej równoważności należy podchodzić z pewną ostrożnością. Jednak pojęcie to dobrze oddaje zasady zawarte w prawie Leibniza.

Jeśli wszystkie obserwacyjnie różne programy mają różne wartości semantyczne, to o ich semantyce mówi się, że jest *zgodna* (ang. *sound*), bo spełnia zasadę:

dla wszystkich kontekstów C, jeśli $||P|| = ||Q||$, to $||C[P]|| = ||C[Q]||$.

A zatem pojęcie identyczności wprowadzone poprzez semantyki zgodne spełnia zasadę nieodróżnialności identycznych.

Semantyka jest *pełna* (ang. *complete*), jeśli każde dwa programy o różnej wartości semantycznej są obserwacyjnie różne (nie są one równoważne), to znaczy:

dla wszystkich kontekstów C, jeśli $||C[P]|| = ||C[Q]||$, to $||P|| = ||Q||$.

Łatwo zauważyć, że semantyki takie spełniają zasadę identyczności nieodróżnialnych.

I na koniec: semantykę nazywa się *w pełni abstrakcyjną*, jeśli jest ona zgodna oraz pełna. Jak widać, wynika z tego, że prawo Leibniza spełniają semantyki w pełni abstrakcyjne i one właśnie powinny służyć do określania równości programów, ponieważ oddają strukturę programów, abstrahując od szczegółów notacyjnych czy implementacyjnych. Warto zauważyć, że wiele semantyk denotacyjnych nie jest w pełni abstrakcyjnych, a wiele se-

mantyk operacyjnych posiada tę własność. Nie można zatem jednoznacznie stwierdzić, że semantyki operacyjne nie mogą służyć do określania równości programów, podczas gdy semantyki denotacyjne są do tego celu odpowiednie. Konieczne jest jeszcze sprawdzenie ich własności logicznych.

Semantyki języków programowania poza określaniem kryteriów równości programów mogą mieć inne jeszcze, liczne zastosowania. Używa się ich między innymi w badaniach własności – a właściwie „zachowań" programów, na podstawie matematycznych własności ich wartości semantycznych. Można w ten sposób badać – na przykład – poprawność programów lub ich części oraz wykorzystywać je do projektowania języków programowania posiadających odpowiednie własności metateoretyczne.

Jedną z cech „dobrego" języka programowania jest jego przejrzystość referencyjna (wzajemna zastępowalność termów stojących za tymi samymi obiektami). Języki powodujące efekty uboczne – umożliwiające na przykład zmianę wartości zmiennych – nie mają tej własności. Przejrzystość referencyjna przyczynia się do lepszej kontroli poprawności programu (ułatwia jej dowodzenie). Umożliwia ona także stosowanie tak zwanego leniwego wartościowania wyniku funkcji, a więc obliczanie tylko tych fragmentów wyniku, które faktycznie są potrzebne innej funkcji lub akcji. Wreszcie, przejrzystość referencyjna (w tym – niezależność od efektów ubocznych) wraz z leniwym wartościowaniem pozwala na obliczanie składowych funkcji niezależnie od siebie, co w naturalny sposób ułatwia obliczenia równoległe[12].

Jeśli chcemy stworzyć język programowania, który jest przejrzysty referencyjnie, to musi to być język niezmieniający wartości zmiennych. Jak zatem powinien on wyglądać? Zmienne w tym języku mogą przyjmować pewne wartości, które nie mogą ulegać zmianie, natomiast podprogramy (pobierające parametry i zwracające wyniki) traktować należy jak funkcje matematyczne. Język programowania jest wówczas referencyjnie przejrzysty. Wymaga to jednak użycia pojęcia funkcji „wyższego rzędu" – argumentami jej mogą być inne funkcje. Istnieją języki programowania dysponujące odpowiednimi narzędziami do wyrażania takich konstrukcji, jak na przykład języki funkcyjne takie jak Haskell czy Lisp, języki nietypowane (Scheme) czy języki typowane (na przykład ML).

I to właśnie ze względu na tę normatywną rolę semantyki w procesie tworzenia nowych języków programowania twierdzi się (Turner, Eden, 2011), że to ona jest centralnym zagadnieniem informatyki – pozwala bowiem, między innymi, na lepsze zrozumienie natury czynności wchodzących w skład procesu tworzenia programów.

---

[12] Więcej szczegółów technicznych znaleźć można np. w książce *Przegląd języków i paradygmatów programowania* (Bylina, Bylina, 2011).

2.1.5. Specyfikacja

Jak wspomniano wcześniej, tworzenie programu komputerowego nie jest prostą czynnością, lecz skomplikowanym procesem. Rozpoczyna się on od sformułowania specyfikacji, to znaczy od określenia, co program powinien „robić". Każde przedstawienie wymagań stawianych przed programem – lub innym artefaktem, na przykład komputerem – nazwać można specyfikacją.

W praktyce specyfikacje wyrażane są w wielu językach i formalizmach, różnią się też poziomem szczegółowości. Często spotyka się specyfikacje urządzeń i programów wyrażone w języku naturalnym (potocznym), ale wielu informatyków twierdzi jednak, że ze względu na jego wieloznaczność i niejasność nie nadaje się on do tego celu i powinien zostać zastąpiony ścisłym językiem formalnym.

W roku 1985 Meyer do specyfikacji programu formatującego fragment tekstu zaproponował właśnie system formalny. Był on stosunkowo prosty, bowiem do zastąpienia języka potocznego formalnym potrzebna była tylko znajomość pojęć: zbioru, ciągu, relacji, funkcji, oraz umiejętność posługiwania się rachunkiem predykatów. Z kolei Turner (2005) pokazał, że specyfikacje funkcyjne sformułowane metodami formalnymi mogą być wyrażone jako relacje w rachunku predykatów. Tworzenie formalnych specyfikacji programów i urządzeń jest jednak kwestią trudną i dyskusyjną.

I tak nie ma zgody wśród badaczy nawet w kwestii tego, czy specyfikacje formalne są wykorzystywane w praktyce tworzenia oprogramowania. Niektórzy (między innymi DeMillo, Lipton, Perlis, 1979) twierdzą, że w praktyce specyfikacje takie tworzone są bardzo rzadko, ponieważ byłyby one bardzo długie, złożone i „dość dziwne". Specyfikacja dla dowolnego rzeczywistego systemu operacyjnego albo kompilatora zajęłaby całe tomy i nikt nie uwierzyłby, że jest ona kompletna. Inni (na przykład Hall, 1990) uważają, że specyfikacje formalne są wykorzystywane w rzeczywistości – nawet przy tworzeniu dużych aplikacji (między innymi dla przemysłu) – oraz że dzięki nim powstają łatwiejsze do zrozumienia i krótsze dokumentacje projektów programów. Anthony Hall twierdzi, powołując się na własne doświadczenia projektowe i programistyczne, że zastąpienie specyfikacji napisanej w języku naturalnym przez specyfikację formalną wiąże się z wieloma praktycznymi korzyściami. Po pierwsze, specyfikacja formalna pomaga sformułować i uściślić wymagania stawiane przed programem, ułatwiając jednocześnie ich zrozumienie – i to zarówno twórcom oprogramowania, jak i jego przyszłym użytkownikom. Po drugie, pozwala na wczesne wykrycie i poprawienie błędów lub nieścisłości w koncepcji działania programu, a co za tym idzie – na podjęcie decyzji o zmianach dotyczących funkcjo-

nalności przed rozpoczęciem jego pisania (prowadzi do zmniejszenia kosztów jego tworzenia). Po trzecie, umożliwia uzasadnienie pewnych własności programu, których nie można udowodnić innym metodami niż formalne. Po czwarte, pomaga ona także w kolejnym etapie tworzenia programu, a mianowicie w jego implementacji. Hall (1990) pisze: „Nasze doświadczenie pokazuje, że łatwiej jest zbudować system ze specyfikacji formalnej, niż używając innych metod" (s. 16).

Co więcej, wbrew powszechnemu przekonaniu, tworzenie specyfikacji formalnych nie wymaga zaawansowanej wiedzy matematycznej. Podstawowym problemem podczas ich formułowania nie jest matematyka, która zazwyczaj nie wykracza poza znajomość teorii mnogości i logiki, ale uchwycenie właściwych powiązań pomiędzy światem rzeczywistym a formalizmem matematycznym.

Wymóg tworzenia specyfikacji formalnych jest dość często akceptowany przez informatyków – niektórzy z nich idą nawet o krok dalej i wymagają, by były one wykonywalne (Agresti, 1986; Fuchs, 1992). Specyfikacje formalne są zazwyczaj rozumiane jako modele koncepcyjne programów, podczas gdy specyfikacje wykonywalne tworzą swego rodzaju modele funkcjonalne. Tworzenie poprawnych programów jest jednym z kluczowych zagadnień informatyki, a specyfikacje wykonywalne mogą znacznie ułatwić to zadanie. Pozwalają one bowiem na wczesne wykrywanie i poprawianie błędów w projektach programów, i co więcej, umożliwiają natychmiastowe sprawdzenie tych fragmentów oprogramowania, których nie można sprawdzić w sposób formalny – na przykład, kontrolę interfejsu użytkownika. Warto w tym miejscu zauważyć, że istnieje wiele wymogów stawianych przed programami komputerowymi, których nie da się sformalizować – wymienić tu można choćby łatwość obsługi. Wymogi te zazwyczaj są dołączane do specyfikacji formalnych w postaci uwag i przypisów, a nie można sprawdzić ich inaczej niż przez przetestowanie, co jest możliwe przed napisaniem programu tylko dzięki zastosowaniu specyfikacji wykonywalnych. Fuchs (1992) uważa, że „[w]ykluczenie wykonywalności z języków specyfikacji oznacza pozbawienie się silnej metody sprawdzania poprawności" (s. 323).

Ponadto specyfikacje wykonywalne mogą służyć jako prototypy, umożliwiające eksperymentowanie z różnymi wymogami stawianymi przed programem, co znakomicie pasuje do jednego z modeli tworzenia oprogramowania – modelu ewolucyjnego. Jest to mianowicie model często stosowany w przypadku, gdy nie można w początkowym etapie tworzenia programu określić, w sposób dokładny i pełny, wszystkich stawianych przed nim wymagań. Wykonywalne specyfikacje mogą wówczas stać się znakomitym narzędziem komunikacji pomiędzy zamawiającym program klientem a two-

rzącym go zespołem programistycznym, umożliwiają bowiem przedstawienie działania programu spełniającego konkretne wymogi.

Pomimo wielu zalet specyfikacji wykonywalnych, nie wszyscy zgadzają się z koniecznością powszechnego ich wykorzystania. Hayes i Jones (1989) uważają, że siła wyrazu języka specyfikacji i jej wykonywalność wykluczają się wzajemnie – a to właśnie siła wyrazu jest w specyfikacji najistotniejsza. Wykonywalność nieuchronnie ogranicza uniwersalność i ekspresyjność języków specyfikacji, wprowadzając do nich szczegóły algorytmiczne konieczne do jej wykonania. Łatwiej jest również sprawdzić, że implementacja odpowiada specyfikacji, gdy specyfikacja ta jest bardziej abstrakcyjna, nie zawiera na przykład struktur danych i detali algorytmicznych występujących w specyfikacjach wykonywalnych. Co więcej, specyfikacji wykonywalnych powinno się unikać, ponieważ mogą one źle wpływać na implementację, w pewien sposób – być może nie wprost – narzucając detale programistyczne i ograniczając inwencję twórców oprogramowania. Autorzy na poparcie swoich tez przytaczają szereg kwestii, które ich zdaniem nie mogą być w sposób właściwy uchwycone przez specyfikacje wykonywalne.

Przykłady te rozważa w swej pracy Fuchs (1992); wykazuje, że siła wyrazu i wykonywalność specyfikacji nie muszą się wykluczać, jeśli specyfikacje pisane są w językach deklaratywnych, na przykład – w językach funkcyjnych (takich jak ML) bądź też logicznych (Prolog). Autor omawia poszczególne problemy wskazane przez Hayesa i Jonesa oraz przedstawia właściwe dla nich specyfikacje wykonywalne, zapisane w logicznym języku LSL[13]. Pokazuje także, w jaki sposób można przeformułować specyfikacje niewykonywalne w języku LSL, dodając kilka elementów konstruktywnych, takich jak reprezentacja zbiorów i ciągów za pomocą list czy też rekurencyjne definicje zbiorów. Fuchs twierdzi, że niewykonywalne specyfikacje, które są konstruktywne, można łatwo „przetłumaczyć" na wykonywalne – prawie bez straty poziomu ogólności (na tym samym poziomie abstrakcji), bez konieczności wprowadzania nowych algorytmów czy też wyboru szczegółów implementacyjnych (które można w każdym momencie zmienić). Jego zdaniem języki logiczne posiadają wystarczającą siłę wyrazu, by używać ich z powodzeniem do tworzenia specyfikacji wykonywalnych praktycznie wszystkich, nawet bardzo złożonych, programów.

Zagadnienia używania w procesie tworzenie programów specyfikacji formalnych, a w szczególności wykonywalnych, oraz istnienia metod automatycznego tworzenia oprogramowania wprost ze specyfikacji generują pytania o różnice pomiędzy specyfikacją programu a nim samym. Jak

[13] Język LSL został przedstawiony przez Lloyda i Topora w pracy *Making Prolog More Expressive* (Lloyd, Topor, 1984).

odróżnić specyfikacje od programów przez nie opisywanych? Istnieją różne odpowiedzi na to pytanie.

Jedno z najczęściej przytaczanych kryteriów odróżniających specyfikacje od programów związane jest z poziomem ich szczegółowości. Program zawiera bardzo szczegółowy opis działania komputera (instrukcja po instrukcji), specyfikacja natomiast opisuje tylko związek pomiędzy wejściem a wyjściem programu (to znaczy, jakie efekty i dla jakich danych wejściowych chcemy otrzymać). Kryterium to jednak nie zawsze da się zastosować w przypadku formalnych specyfikacji zapisanych z użyciem specjalnych języków o poziomie szczegółowości zbliżonym do poziomu programów (na przykład – napisanych z użyciem funkcyjnych języków programowania). Jeśli zatem uznać kryterium poziomu szczegółowości za niewystarczające dla odróżnienia specyfikacji od programów, można szukać innego – analizując na przykład ich funkcję.

Często twierdzi się, że zadaniem specyfikacji jest opisanie artefaktu, który ma powstać. Specyfikacje mają wtedy charakter deskryptywny – w odróżnieniu od programów, które są imperatywne. Jest to jednak rozróżnienie właściwe tylko w przypadku tak zwanego programowania imperatywnego – program jest wtedy listą rozkazów, jakie komputer ma w określonej kolejności wykonać. Jest on tylko jednym z wielu obowiązujących paradygmatów programowania. W przypadku innych języków, na przykład logicznych, funkcyjnych czy zorientowanych obiektowo, trudno jest mówić o „ciągu instrukcji". Właściwsze wydaje się twierdzenie, że programy takie składają się z ciągów definicji. Programy napisane w tych językach nie mają wyraźnie widocznego charakteru imperatywnego. Trzeba tu nadmienić, że nie wszystkie specyfikacje mogą być rozumiane jedynie jako stwierdzenia dotyczące relacji pomiędzy wejściem i wyjściem, gdyż zawierają one dodatkowe wymagania związane – na przykład – z rodzajem instrukcji, jakich należy użyć w programie. A zatem nie wszystkie programy są imperatywne i nie wszystkie specyfikacje są deskryptywne, a co za tym idzie, kwestia deskryptywne czy imperatywne nie jest odpowiednim kryterium je odróżniającym. Spróbujmy zatem poszukać innego rozwiązania.

Można twierdzić, że program komputerowy – w przeciwieństwie do specyfikacji – może zostać skompilowany i wykonany. Nie jest jednak jasne, co należy rozumieć pod pojęciem możliwości skompilowania? Jeśli chodzi o fizyczne istnienie kompilatora – status obiektu może zmieniać się w czasie. Obecnie dany obiekt jest specyfikacją, gdyż nie istnieje dla niego kompilator, jutro natomiast może on powstać. Zdarza się tak, że dzisiejsze języki specyfikacji stają się z czasem językami programowania, tak jak to się stało w przypadku języków funkcyjnych. Możemy zatem twierdzić, że języki programowania to języki, które posiadają implementację, języki specyfikacji

natomiast jej mieć nie muszą. Dzieje się tak dlatego, że w tych ostatnich dopuszcza się wyrażenia, które – w sensie Turinga – nie są obliczalne[14]. Jednak, jak wspomniano wcześniej, istnieją także specyfikacje wykonywalne. Warto zastanowić się nad różnorodnymi celami tworzenia specyfikacji i programów – nad intencjami ich twórców (Fuchs, 1992). Otóż specyfikacje mają tylko oddawać funkcje i zachowanie systemu w sposób jak najbardziej ogólny i czytelny, bez względu na jego wydajność, a ona jest właśnie jedną z najistotniejszych własności programu, uzyskiwaną często nawet kosztem jego czytelności.

Rozróżnienie specyfikacji i programu nie polega zatem na ocenie poziomu szczegółowości, użytego do ich zapisu języka, czy też istnienia implementacji. Specyfikacja, w odróżnieniu od programu, jest nie tylko opisem działania artefaktu, bowiem wprowadza ona również kryteria poprawności tegoż działania. Pełni zatem funkcję normatywną – mówi programiście lub inżynierowi, co ten powinien zbudować, oraz informuje o tym, czy wykonał on zadanie dobrze. Złe funkcjonowanie nie jest bowiem własnością samej maszyny, lecz pojęciem wprowadzanym *implicite* przez jej specyfikację. Czym jest tak pojmowana specyfikacja?

Jeśli traktujemy specyfikacje jako opisy artefaktów, można wówczas twierdzić, że są one po prostu definicjami nowych urządzeń, programów lub całych systemów. Jakiego rodzaju są to definicje? Raymond Turner (2011) twierdzi, że specyfikacje są definicjami warunkowymi nowych artefaktów, gdyż wprowadzają nowe terminy – o znaczeniu określonym w danym kontekście.

Powstaje jednak pytanie, czy każda definicja warunkowa jest specyfikacją. W myśl przyjętego wcześniej założenia, definicja funkcjonuje jako specyfikacja tylko wtedy, gdy oprócz opisu obiektu podaje ona także kryterium jego poprawnego działania. Można odwołać się w tym miejscu do rozróżnienia kontekstu odkrycia i kontekstu uzasadnienia w filozofii nauki. W przypadku specyfikacji artefaktu istnieje istotna różnica pomiędzy procesem jego konstruowania – a uzasadnieniem poprawności tego procesu. Podsumowując, jeśli definicja ma siłę normatywną w procesie konstrukcji artefaktu, to jest ona specyfikacją, a nie zwykłą definicją. A zatem – nie każda definicja warunkowa w informatyce może być traktowana jak specyfikacja[15].

---

[14] Możemy chcieć wyrazić w specyfikacji własność programu, której nie da się zapisać w jego ciele, na przykład możemy nalegać, by funkcja była totalna.

[15] Jeśli rozumiemy specyfikacje jako definicje, to bardzo trudno jest mówić o ich poprawności, ponieważ definicje nie mogą być poprawne lub niepoprawne w klasycznym rozumieniu terminu „poprawny". Wydaje się, że bardziej odpowiednie jest mówienie o adekwatności definicji, a tym samym o adekwatności specyfikacji.

Można pójść o krok dalej i twierdzić, że specyfikacja jest odpowiednikiem teorii naukowej, którą informatycy poddają sprawdzeniu poprzez skonstruowanie artefaktu przez nią opisywanego. Czy analogia ta jest trafna? Powszechnie przyjmuje się, że teorie naukowe powinny umożliwiać przewidywanie i wyjaśnianie zjawisk zachodzących w świecie. Specyfikacje, co prawda, mogą mieć logiczne konsekwencje i można je traktować jako predykcje na temat własności konstruowanego urządzenia, jednak to, co testuje się eksperymentalnie, to nie specyfikacja, lecz sam artefakt. Co więcej, specyfikacje nie są opisami świata – nie podają bowiem wiedzy na temat tego, jaki on jest, nie są też wyjaśnieniami naukowymi i nie przewidują nawet, czy artefakt będzie działał poprawnie; opisują tylko znaczenie „poprawnego działania". Zatem specyfikacje nie są teoriami naukowymi w klasycznym ich rozumieniu.

Ważną cechą specyfikacji programów i urządzeń jest to, że mogą one podlegać zmianom w czasie. Na przykład – gdy klient zmieni zdanie co do wymagań bądź też gdy okaże się, że zbudowanie artefaktu nie jest możliwe (chociażby ze względu na ograniczenia fizyczne, nieuwzględnione wcześniej przez projektanta). W takich przypadkach specyfikację porzuca się – lub co najmniej modyfikuje. Co więcej, nie zawsze artefakt spełnia swoją specyfikację. Są jednak takie wymagania zawarte w specyfikacji, z których można zrezygnować, oraz takie, które są niezbędne. Decyduje o tym rozróżnieniu *funkcja*, jaką ma pełnić opisywany przez nią artefakt. Specyfikacje traktować można zatem jako opisy funkcjonalne artefaktów, a nie jako definicje czy teorie naukowe.

Opisy funkcjonalne zawsze odpowiadają na pytanie, do czego służy dany obiekt, lecz tylko niektóre z nich podają również sposób jego konstrukcji (w takich przypadkach specyfikacja informuje nie tylko o tym, co zbudować, ale również, jak to zrobić). Opisu sposobu konstrukcji obiektu, jeśli taki istnieje, nie można jednak traktować jako części specyfikacji, a w szczególności – specyfikacji funkcjonalnej.

Gdyby traktować specyfikacje jako opisy funkcjonalne, stajemy wobec pytania, czy są to pojęcia różne – a jeśli tak, to jak je rozróżnić. Specyfikacja i opis funkcjonalny mogą mieć bowiem tę samą formę i zbliżony poziom szczegółowości oraz mogą także być opisane podobnym językiem (formalnym lub nie). Jedyna istotna różnica pomiędzy nimi związana jest z czasem ich powstania (Turner, 2011). Jeśli opis powstał przed konstrukcją obiektu, na przykład – przed napisaniem programu lub zbudowaniem maszyny – to jest to specyfikacja (posiada bowiem normatywny charakter), w przeciwnym przypadku – to tylko opis funkcjonowania.

Różnorodność przytoczonych powyżej poglądów dotyczących natury specyfikacji i jej roli w procesie tworzenia programów wskazuje, że dogłębne jej zrozumienie wymaga jeszcze głębszej analizy filozoficznej, w tym –

pojęciowego wyjaśnienia. Jest to tym ważniejsze, że analiza ta może mieć wpływ na projektowanie przyszłych języków programowania i języków specyfikacji. Turner (2013a) twierdzi, że „[…] natura specyfikacji jest fundamentalnym zagadnieniem dla koncepcyjnych podstaw informatyki".

Nie jest ona jednak jedynym ważnym i interesującym z filozoficznego punktu widzenia etapem tworzenia oprogramowania. Równie ciekawa jest implementacja.

2.1.6. Implementacja

Pojęcie implementacji jest w informatyce terminem wieloznacznym, prawie nigdy nie definiowanym, ale dobrze rozumianym. Dlatego też jest ciekawym tematem do analizy filozoficznej.

Pojęcie to można rozumieć bardzo szeroko jako realizację specyfikacji (Turner, 2013), najczęściej jednak termin ten spotyka się w kontekście zapisu algorytmu w języku programowania. Mówi się wówczas, że (1) program jest implementacją algorytmu. Nie jest to jednak jedyne użycie tego terminu. Pojawia się on również w innych kontekstach, na przykład: (2) programy są implementowane w języku maszynowym, bądź też (3) konkretne struktury danych (na przykład tablice) są implementacją abstrakcyjnych typów danych (na przykład stosów). Czy jest zatem implementacja relacją pomiędzy czymś „konkretnym" a abstrakcją, czy też może ona łączyć dwa pojęcia abstrakcyjne? Jeśli jest relacją między czymś „konkretnym" a abstrakcją, to jakiego typu jest to relacja?

Przyjrzymy się na początek implementacji programów w rozumieniu (2). Eden i Turner (2006) opisując kompilację (tłumaczenie kodu źródłowego zapisanego w języku programowania wysokiego poziomu na język konkretnej klasy maszyn), twierdzą, że proces ten jest odwzorowaniem typu jeden do wielu. Zatem implementacja w drugim znaczeniu jest taką właśnie relacją. Eden i Turner argumentują przy tym, że tłumaczenia z konkretnego języka programowania do języków rodziny procesorów Intel mogą się różnić w zależności od: docelowego języka maszynowego, dystrybutora i wersji kompilatora. Na przykład w języku C zdefiniowano typ int jako „zależny od implementacji". Pozwala to sprzedawcom kompilatora C dla 16-bitowych procesorów reprezentować liczby całkowite na 2 bytach, a dla procesorów 32-bitowych – na 4 bytach. A zatem implementacja rozumiana jako kompilacja (lub interpretacja) kodu źródłowego języka programowania wysokiego poziomu na kod maszynowy jest relacją [typu] jeden do wielu.

Szczegółową analizę pojęcia implementacji przedstawił Rapaport (1999, 2005). Podaje on wiele jej przykładów, zarówno z dziedziny informatyki, jak i spoza niej, między innymi:

Tabela 3. Przykłady implementacji

Dziedzina semantyczna		Dziedzina syntaktyczna
1. Program komputerowy	jest implementacją	algorytmu
2. Proces obliczeniowy	jest implementacją	programu komputerowego
3. Struktura danych	jest implementacją	abstrakcyjnej struktury danych
4. Wykonanie utworu	jest implementacją	jego partytury
5. Dom	jest implementacją	jego projektu
6. Model teoriomnogościowy	jest implementacją	teorii formalnej

Źródło: Rapaport, 1999, s. 2

Rapaport (1999) uważa, że we wszystkich tych przypadkach implementacja jest interpretacją semantyczną oraz że: „Interpretacja semantyczna wymaga dwóch dziedzin (ang. *domain*) i jednej relacji: *dziedziny syntaktycznej*, charakteryzowanej przez reguły, manipulowania na symbolach […]; *dziedziny semantycznej*, charakteryzowanej podobnie, oraz relacji interpretacji semantycznej, która odwzorowuje jedną w drugą" (s. 1). Twierdzi on jednocześnie, że nie ma żadnych wewnętrznych cech czyniących obiekt dziedziną syntaktyczną lub semantyczną. W przedstawionych w powyższej tabeli przykładach program komputerowy jest w jednym ujęciu dziedziną semantyczną (program jest implementacją algorytmu), a w innym – syntaktyczną (proces obliczeniowy jest implementacją programu). Tym, co powoduje, że ten sam obiekt może występować w dwóch różnych „rolach", jest asymetryczność relacji interpretacji: dziedzina semantyczna (nazywana przez autora „Implementacją") implementuje dziedzinę syntaktyczną („Abstrakcję"). Zatem Implementacja jest interpretacją semantyczną Abstrakcji.

Pełne wyjaśnienie pojęcia implementacji wymaga jednakże użycia jeszcze jednego terminu – nośnika (medium) implementacji. Jest ona zatem relacją ternarną:

I jest implementacją *A* w medium *M*,

gdzie *I* jest dziedziną semantyczną, A jest Abstrakcją, natomiast *M* może być zarówno obiektem konkretnym (fizycznym), jak i abstrakcyjnym. Na przykład strukturę danych, jaką jest stos, można zaimplementować jako listę[16], z kolei listę tę implementuje się w języku programowania (na przykład w Pascalu) – otrzymując kod źródłowy. Implementacja kodu źródłowego (jego kompilacja) z kolei daje kod w jakimś języku maszynowym – i dalej, tak zapisany program implementuje się w rzeczywistym komputerze. W tym „ciągu" implementacji jej nośniki rozpoczynają się od najbardziej

[16] Ten rodzaj implementacji omówimy bardziej szczegółowo poniżej.

abstrakcyjnych (struktur danych) i stają się coraz bardziej konkretne (ostatnim ogniwem jest obiekt fizyczny – komputer). Można zatem stwierdzić, że implementacja jest „realizacją" pewnego obiektu w jakimś nośniku – może on być fizyczny lub abstrakcyjny. Czyli: istnieją dwa typy implementacji – konkretna i abstrakcyjna. Rapaport (1999) pisze: „Implementować to konstruować coś, bez surowców w ręce, co ma własności Abstrakcji; może to być również znalezienie odpowiednika, który ma te własności. *Oba zadania są semantyczne*" (s. 3). Uzasadnia swoje ostatnie stwierdzenie tym, że dowolna odpowiedniość pomiędzy dwiema dziedzinami, w której jednej używa się, by rozumieć lub zinterpretować drugą, jest odpowiedniością semantyczną. Poprawne jest zatem mówienie o semantyce w odniesieniu do implementacji. W podobnym kontekście pojawia się ten termin u Hayesa (1988): „Istnieje zatem *luka semantyczna* (ang. *semantic gap*) pomiędzy wysokiego poziomu specyfikacją problemu i zbiorem instrukcji maszynowych, które go implementują, luka, którą musi wypełnić kompilator" (s. 209).

Tak rozumiane pojęcie implementacji ułatwia wyjaśnienie natury symulacji komputerowych. Wspomnianą lukę semantyczną można bowiem wytłumaczyć jako złożenie następujących relacji:

A. Program zapisany w języku programowania wysokiego poziomu jest interpretowany semantycznie przez obiekty świata rzeczywistego (na przykład rekordy[17] w Pascalu mogą reprezentować studentów).
B. Program ten jest kompilowany do implementacji – programu w języku maszynowym.
C. Implementacja w języku maszynowym jest z kolei interpretowana semantycznie przez bity w komputerze.

Można rozumieć program zapisany w Pascalu jako matematyczny model obiektów rzeczywistych (studentów), a program zapisany w języku maszynowym jako model bitów w komputerze (również obiektów świata rzeczywistego). Luki semantycznej należy szukać pomiędzy obiektami świata rzeczywistego (studentami) a bitami w komputerze. Relacja ta jest symulacją:

---

[17] Rekord to złożony typ danych (występujący m.in. w Pascalu), który służy do przechowywania kilku zmiennych związanych ze sobą w ścisły sposób (np. opisujących jeden obiekt). Na przykład:

```
type tStudent = Record
   Imie : String;
   Nazwisko : String;
   Wiek : Byte;
   Kierunek_studiów: String;
   Rok_studiów: Byte;
   End;
```

bity są symulacjami studentów – a symulacja jest rodzajem implementacji. Bity są bowiem komputerową implementacją studentów (jej nośnikiem jest komputer).

Pojęcie implementacji pojawia się jednak w informatyce częściej w innym kontekście niż symulacje – w kontekście realizacji abstrakcyjnych typów danych, czyli w rozumieniu (3).

Przyjrzyjmy się dla przykładu możliwym realizacjom jednego z abstrakcyjnych typów danych – stosu. Stos (ang. *stack*) jest liniową strukturą danych, w której dane dokładane są zawsze na wierzchu stosu – nazywamy go wierzchołkiem – i z niego są pobierane. Jest to zatem struktura typu LIFO (*Last In, First Out*), to znaczy: ostatni wchodzi, pierwszy wychodzi. Ideę stosu danych można zilustrować stertą książek ułożonych jedna na drugiej. Kolejne książki dodaje się wówczas do stosu, kładąc je na jego wierzchołku – i stamtąd również się je zdejmuje. Aby uzyskać dostęp do elementów stosu poniżej wierzchołka, trzeba najpierw zdjąć po kolei to, co jest nad nimi. Podstawowe operacje, jakie można wykonać na takiej strukturze danych, to: `push(obiekt)` – czyli odłożenie obiektu na stos oraz `pop(obiekt)` – ściągnięcie obiektu ze stosu i zwrócenie jego wartości. A zatem aby móc używać stosu w Pascalu (w którym taki typ danych nie jest typem standardowym), należy znaleźć metodę przedstawienia tak ustrukturyzowanych danych (w tym dostęp do wierzchołka stosu) oraz – zaproponować metody realizacji operacji `push` i `pop`.

Jedną z możliwości jest zaimplementowanie stosu *s* w postaci tablicy (jednowymiarowej tabeli) *A*, w której umieszczamy kolejne elementy stosu: *A*`[0],…,` *A*`[n]`. Wierzchołkiem jest wówczas element `A[0]`. Stanowi to jednak pewne uproszczenie, ponieważ stos jako abstrakcyjny typ danych nie jest ograniczony, natomiast tabela ma skończoną długość (można w niej umieścić tylko *n* elementów). Operacja `push(s,i)` jest wtedy procedurą dwuargumentową, „dokładającą" element *i* do stosu *s*, to znaczy modyfikującą stos następująco: `A[0]:=i` (pierwszym elementem – wierzchołkiem – staje się element dokładany, czyli *i*) oraz `A[j]:=A[j-1]` dla `j=1,…,n` (pozostałe elementy stosu są „przesuwane" w dół). Operacja `pop(s)` jest z kolei jednoargumentową funkcją zwracającą element znajdujący się na pierwszym miejscu listy `(A[0])`, a ponadto powodującą usunięcie pierwszego elementu tablicy i przesunięcie pozostałych elementów „w górę" (to znaczy `A[j]:=A[j+1]`)[18]. Nie jest to jednak jedyna możliwa realizacja stosu w Pascalu.

---

[18] Oczywiście należy wtedy również określić szczegóły implementacyjne np. co zrobić, gdy stos się przepełnia, tzn. gdy próbujemy dołożyć element (n + 1)-szy do tabeli długości n.

Strukturę stosu można przedstawić za pomocą innej realizacji – nazywanej listą. Lista sama w sobie jest abstrakcyjnym typem danych służącym do reprezentacji elementów ułożonych w porządku liniowym. Rozróżniane są dwa jej podstawowe typy: jednokierunkowe i dwukierunkowe. Każdy element listy składa się z co najmniej dwóch pól: danych oraz – pola wskazującego na następny jej element (w przypadku listy jednokierunkowej). W przypadku listy dwukierunkowej natomiast – każdy jej element zawiera także pole wskazujące na poprzedni element (elementy składają się wtedy z co najmniej trzech pól). Pośród podstawowych operacji wykonywanych na liście *l* warto wyróżnić następujące: `first(l)` – funkcja zwracająca pierwszy element listy; `rest(l)` – funkcja zwracająca listę zawierającą wszystkie elementy *l* z wyłączeniem pierwszego; `list(i,l)` – zwiększenie (przedłużenie) listy *l* poprzez dodanie do niej na początku elementu *i*. Operacje te pozwalają na zdefiniowanie stosu *s* jako listy jednokierunkowej *l* z wierzchołkiem `first(l)` oraz operacjami `push(s,i):=list(l,i)` oraz `pop(s)`, która definiuje listę *l* jako `rest(l)` oraz zwraca wierzchołek (to znaczy `first(l)`). Warto zauważyć, że lista jednokierunkowa może zostać przedstawiona (zaimplementowana) w Pascalu jako tablica dwuwymiarowa z odpowiednimi operacjami. Zatem stos można zaimplementować w nośniku (medium) innego abstrakcyjnego typu danych, to znaczy implementacja nie musi być „konkretna", gdyż abstrakcyjne struktury danych można implementować w innych strukturach abstrakcyjnych.

W powyższym przykładzie „nośnikiem" jest lista, a „język programowania" składa się z operacji na tej liście. Implementacja abstrakcyjnego typu danych składa się zatem z jego reprezentacji oraz – z programów realizujących operacje na tym typie. Elementy jednej dziedziny oraz operacje na niej są wówczas odwzorowywane w innej dziedzinie i innych predykatach, a zatem implementacja jest interpretacją semantyczną.

W ten sposób zdefiniowaną implementację można próbować porównać z innymi typami relacji: indywidualizacją, instancjacją, redukcją i superweniencją (Rapaport, 1999). Jednak wszystkie one wydają się „słabsze" niż implementacja, choć są blisko z nią związane. Zatem „[j]edyną najlepszą 'interpretacją' implementacji wydaje się być interpretacja semantyczna: *I* jest implementacją *A* w medium *M* wtedy i tylko wtedy, gdy *I* jest interpretacją semantyczną lub modelem *A*, gdzie *A* jest pewną dziedziną syntaktyczną oraz *M* jest dziedziną semantyczną" (Rapaport, 1999, s. 19).

Warto także skupić się na następującym zagadnieniu: Czy każdy przykład implementacji można wyjaśnić w terminach interpretacji semantycznej? Czy pojęcie implementacji jest synonimem semantyki, czy też pojęcia te różnią się w jakiś sposób?

Można przyjąć, że każda semantyka powinna dostarczać kryteriów dla określenia poprawności stosowania danego terminu lub wyrażenia w danym kontekście (Turner, 2013). Turner twierdzi, że ujęcie semantyczne musi ustalać, co oznacza poprawne użycie wyrażenia, ponieważ tylko wtedy można ustalić na przykład, że użycie pewnego wyrażenia jest poprawne w stosunku do jednych obiektów, a niepoprawne w odniesieniu do innych. Czy przedstawione powyżej przykłady implementacji, jako interpretacji semantycznej, spełniają ten wymóg?

Rozpocznijmy od jednego z podstawowych przykładów implementacji – od implementacji jednego języka w innym. Z tym rodzajem implementacji mamy do czynienia na przykład podczas kompilacji programu napisanego w języku programowania wysokiego poziomu na język maszynowy. Zarówno dziedziną semantyczną, jak i syntaktyczną są wówczas języki. Niestety, implementacja taka nie wprowadza kryterium poprawności – chyba że język docelowy posiada już ustaloną semantykę. W przeciwnym przypadku „klasyczny" przykład implementacji nie spełnia omawianego wymogu. Można oczywiście wprowadzić kryterium poprawności poprzez dodanie do języka docelowego semantyki – bądź to w postaci nieformalnej, bądź też w sposób matematyczny. Jednakże samo tłumaczenie pomiędzy językami, jako interpretacja semantyczna, w ogólności takiego kryterium nie wprowadza, a więc nie spełnia stawianego semantykom wymogu.

Powróćmy teraz do przedstawionego powyżej przykładu implementacji stosu (abstrakcyjnej struktury danych) w postaci tablicy. Dziedziną syntaktyczną jest wtedy stos, a semantyczną – tablica. Co należy rozumieć przez stwierdzenie, że implementacja ta wprowadza kryterium poprawności? Tablica (dziedzina semantyczna) jako implementacja stosu powinna wprowadzać kryterium poprawnego użycia terminu „stos". Jednakże to nie tablica wprowadza kryterium poprawności aksjomatów opisujących stos, lecz na odwrót – można je traktować jako opis kryteriów poprawnej implementacji stosu w postaci tablicy. A zatem tablica, jako implementacja stosu, nie wprowadza kryterium jego poprawności, bowiem jest dokładnie odwrotnie!

Innym przykładem implementacji jest związek pomiędzy maszyną abstrakcyjną a jej fizyczną realizacją. Do dziedziny semantycznej należy wówczas maszyna fizyczna, a do syntaktycznej – abstrakcyjna – i określa się, że pierwsza jest implementacją drugiej. Mamy wtedy do czynienia z sugestią, że to maszyna fizyczna powinna wprowadzać interpretację semantyczną dla obiektu abstrakcyjnego. Istnieją jednak pewne argumenty przeciwko takiemu poglądowi. Powszechne jest twierdzenie, że to pojęcie funkcji maszyny pozwala na odróżnienie poprawnego jej działania od niepoprawnego[19], a więc

[19] Por. paragraf 1.1.1.

to maszyna abstrakcyjna wprowadza kryterium poprawności dla maszyny fizycznej, a nie na odwrót. Fizyczny artefakt nie może przecież spełniać wprowadzonego przez siebie kryterium poprawności.

Z przedstawionych powyżej przykładów wynika, że przyjmując wymóg wprowadzania przez semantykę kryterium poprawności, powinniśmy uznać, iż niektóre implementacje nie są interpretacjami semantycznymi – gdyż nie dostarczają one takiego kryterium. Można wówczas twierdzić, że rozumienie implementacji jako interpretacji semantycznej stanowi błędne utożsamienie dwóch istotnie różnych pojęć, albowiem – wbrew opinii Rapaporta – nie każdy związek pomiędzy dwiema dziedzinami, w której jedna jest używana dla rozumienia lub zinterpretowania drugiej, jest interpretacją semantyczną[20].

Przedstawione powyżej rozważania dotyczące implementacji pokazują, że pojęcie to nie jest do końca zdefiniowane i że nie ma zgody co do jego natury. Jest to jedno z tych pojęć informatyki, które wymagają głębszej analizy filozoficznej. Jednakże nie można ograniczać rozważań filozoficznych związanych z programami komputerowymi i procesem ich tworzenia tylko do zagadnień ontologicznych. Równie ciekawe i ważne, także z praktycznego punktu widzenia, są kwestie epistemologiczne.

2.2. Epistemologia

Z pojęciem programu komputerowego ściśle związane jest zagadnienie metod badania jego cech, w tym – poprawności. Tworzenie wiarygodnych, niezawodnych systemów komputerowych jest jednym z najważniejszych zadań informatyków. Wydaje się jednak, że określenie dopuszczalnych metod badania programów, w tym sprawdzania ich poprawności, jest zagadnieniem budzącym najwięcej kontrowersji zarówno wśród informatyków, jak i wśród filozofów.

Oczywiście przyjęcie poglądów ontologicznych dotyczących programów komputerowych determinuje związane z nimi poglądy epistemologiczne. Zwolennicy traktowania programów jako obiektów matematycznych postulują używanie do ich badania jedynie metod formalnych. Idea ta jest ściśle związana z *paradygmatem matematycznym* w informatyce, rozumianym jako twierdzenie, że informatyka jest kolejną gałęzią matematyki. Sympatycy traktowania informatyki jako dyscypliny inżynieryjnej, *paradygmatu technokratycznego*, dopuszczają natomiast używanie w procesie weryfikacji

[20] Oczywiście można odrzucić tę argumentację, odrzucając wymóg wprowadzania kryterium poprawności dla semantyki.

programów metod inżynieryjnych, w tym testowania[21]. Przyjmowanie różnych sposobów badania własności programów związane jest z różnym rozumieniem terminu „poprawny program".

2.2.1. Poprawność programów

Wielu informatyków i matematyków prowadziło – i nadal prowadzi – badania związane z określeniem metod sprawdzania poprawności programów komputerowych. Okazuje się jednak, że „poprawność programu" jest terminem wieloznacznym i że nie ma powszechnej zgody co do jego interpretacji.

Jak już wcześniej wspomniano, zarówno samo słowo „program", jak i jego naturę rozumieć można na wiele sposobów, prowadzących do różnych interpretacji terminu „poprawny". Przez pojęcie „poprawny program" rozumie się najczęściej: albo 1) program zgodny ze specyfikacją, albo 2) działający w taki sposób, w jaki powinien, co oznacza w większości przypadków[22], że rozwiązuje on problem, dla którego został stworzony. Niektórzy badacze sugerują wprowadzenie rozróżnienia pomiędzy tymi dwoma znaczeniami – a to poprzez używanie różnych terminów. *Poprawność*, w wąskim znaczeniu technicznym, można rozumieć jako relację pomiędzy programem a specyfikacją, natomiast *wiarygodność* określać może funkcjonowanie programu w rzeczywistym świecie (Smith, 1985).

Zwolennicy rozumienia programów jako bytów matematycznych, myśląc o poprawności w znaczeniu 1), uważają, że aby wykazać poprawność działania programu, należy stworzyć formalny dowód jego zgodności ze specyfikacją. Przy czym przez „dowód" rozumie się wywód formalny podobny do tych, które podają matematycy. Oczywiście aby taki dowód mógł powstać, konieczne jest wcześniejsze stworzenie specyfikacji programu w postaci formalnej. Nie twierdzi się przy tym, że dowody muszą być tworzone przez człowieka, lecz dopuszcza się stosowanie programów weryfi-

---

[21] Współcześnie w informatyce wyróżnić można trzy podstawowe paradygmaty: 1) paradygmat matematyczny, nazywany przez niektórych autorów paradygmatem racjonalnym, 2) paradygmat naukowy definiujący informatykę jako naukę przyrodniczą opartą na eksperymencie, 3) paradygmat technokratyczny traktujący informatykę jako dziedzinę inżynieryjną. Więcej na temat paradygmatów informatyki znaleźć można na przykład w pracach *Paradygmaty informatyki* (Bondecka-Krzykowska, 2010) i *Three Paradigms of Computer Science* (Eden, 2007).

[22] Nie jest do końca jasne, czy wszystkie problemy komputerowe można uznać za „rozwiązujące problemy". Na przykład niektóre z nich (systemy operacyjne) funkcjonują tylko po to, by umożliwić działanie innym programom.

kujących – poprzez zastosowanie odpowiednich systemów formalnych tworzą one tak zwane weryfikacje. Zdaniem zwolenników tej metody, tylko stworzenie formalnego dowodu gwarantuje poprawność sprawdzanego programu. Nie jest to jednak pogląd powszechny.

Niektórzy informatycy często sądzą, że tworzone przez nich programy są o tyle „ważniejsze" od obiektów materialnych, że powinny być doskonałe oraz że wystarczy energii na to, by takimi je uczynić. DeMillo, Lipton i Perlis (1979) uważają, że: 1) programy nie muszą być doskonałe, 2) nie powinno się marnować czasu i energii na próby ich skonstruowania. Nie można logicznie wydedukować, że most będzie stał, a samolot latał. A więc należy dokonać ostrego rozróżnienia pomiędzy *wiarygodnością* programu a jego *doskonałością*. Piszą oni (DeMillo, Lipton i Perlis, 1979):

> W praktyce programowania nie można dopuścić, by weryfikowalność odsunęła w cień wiarygodność. Naukowcy nie powinni mylić modeli matematycznych z rzeczywistością – weryfikacja nie jest niczym więcej niż modelem wiarygodności. Weryfikowalność nie jest, i nie może być, dominującym podejściem do projektowania oprogramowania (s. 279).

Programy nie mogą być weryfikowane, ponieważ nie można logicznie udowodnić, że system przyczynowy (ang. *causal system*) nie zawiedzie, można jedynie udowodnić poprawność algorytmów wykorzystanych przy tworzeniu programu (Fetzer, 1988). Co więcej, w procesie tworzenia programów mamy do czynienia nie z samą rzeczywistością, lecz tylko z jej modelami (Smith, 1985). Modele nigdy nie są i nie mogą być w pełni adekwatne, tak więc nie ma sensu pytać o poprawność programu w odniesieniu do świata rzeczywistego. Komputery znajdują się „w dziwnym położeniu" – opierając się w swym działaniu na modelach (programach) modeli świata, a działając w rzeczywistym świecie. Istnieje luka pomiędzy światem a jego modelami – dlatego też nie można udowodnić w sposób formalny poprawnego działania programów. Stosowniejsze jest w tym przypadku używanie terminu „wiarygodny" – stosowanego również w odniesieniu do ludzi. Bardzo rzadko mówi się o ludziach, że są „poprawni" – ocenia się raczej ich zachowania w konkretnych sytuacjach. Nie można spodziewać się czegoś więcej w odniesieniu do programów. Można jednak twierdzić, to że system komputerowy jest wiarygodny, a do stwierdzenia tego faktu potrzebna jest obserwacja jego zachowania (wykonania). Sam termin „wiarygodność" sugeruje już właściwe metody badania programów. Smith (1985) pisze:

> [...] weryfikacja programów nie jest jedyną, ani nawet najpowszechniejszą, metodą zdobywania pewności co do tego, że system komputerowy będzie działał właściwie. O programach mówi się, że są akceptowalne, a są one dopuszczane

> do użycia nie dlatego, że dowiedziono ich „poprawności", ale dlatego, że w dłuższym czasie okazały się stosunkowo wiarygodne podczas działań, do których zostały przeznaczone. Jako część tego doświadczenia dopuszczamy niepowodzenia: zawsze musi być miejsce na awarie. Oczywiście nikt nie zaakceptuje programu bez przetestowania: dowód poprawności jest w najlepszym razie dodatkowym zabezpieczeniem, nie zastępuje jednak rzeczywistego życiowego doświadczenia (s. 20).

Zatem zdaniem Smitha wiarygodność jest relacją pomiędzy programem i jego funkcjonowaniem w rzeczywistym świecie, którą należy badać poprzez testowanie, poprawność natomiast można rozumieć jako relację pomiędzy programem a modelem problemu wyrażonym w terminach formalnej specyfikacji.

Wielu informatyków twierdzi podobnie: podczas tworzenia rzeczywistych systemów komputerowych należy porzucić pomysł sprawdzania ich poprawności w sposób absolutny (metodami dowodu formalnego) i skoncentrować się na testowaniu, jako sposobie sprawdzania ich wiarygodności.

2.2.2. Testowanie programów

Testowanie programów polega na ich uruchomieniu i obserwacji działania. Niezwykle istotny jest przy tym właściwy dobór danych wejściowych. Oczywiście w przypadku większości programów, nawet tych prostych, nie jest możliwe sprawdzenie ich działania dla wszystkich możliwych danych początkowych. Na przykład nie można sprawdzić działania programu dodawania dwóch liczb rzeczywistych dla wszystkich możliwych „wejść" (wszystkich par liczb rzeczywistych), ponieważ zbiór liczb rzeczywistych jest nieskończony. Zatem jedną z ważniejszych kwestii metodologii testowania jest wybór, między innymi za pomocą analizy statystycznej, reprezentatywnego zbioru danych początkowych, który zminimalizuje możliwość pominięcia podczas testowania tych „wejść", dla których program nie działa prawidłowo. Jednak nawet jeśli zbiór danych wejściowych zostanie wybrany zgodnie z obowiązującymi zasadami, to nigdy nie można mieć pewności, jak działałby program w przypadkach nieprzetestowanych. E.W. Dijkstra (1989) twierdzi, że „[…] testowanie programu może przekonująco pokazać obecność błędów, ale nigdy nie pokaże ich braku" (s. 1401).

Jest to jedna z najczęściej cytowanych uwag dotyczących testowania; celem jej autora było wskazać, że testowanie jest działaniem pozbawionym sensu. Należy jednak pamiętać, że pomimo tego, iż testowanie nie może udowodnić braku błędów w programie, to może ono zwiększyć przekonanie

co do jego poprawnego działania[23], co jest całkowicie zgodnie z metodologią nauk eksperymentalnych. Nie podważa to zatem celowości testowania jako metody badania programów.

Testowanie w informatyce ma wiele zastosowań. Najczęściej wspomina się o nim w kontekście sprawdzania poprawności programów. Jednak ze względu na wspomnianą wcześniej wieloznaczność terminu „poprawny program" warto przyjrzeć się różnym znaczeniom terminu „testowanie" w odniesieniu do programów. Za kryterium przyjmiemy tu cel, w jakim przeprowadza się testy.

I tak Timothy Colburn (2000) przedstawia kilka możliwych znaczeń terminu „testowanie" w informatyce. Po pierwsze, jeśli przez poprawny program rozumie się ten, który jest zgodny ze specyfikacją, to uruchamiając go podczas testowania, chcemy sprawdzić, czy jego wykonanie zgadza się ze specyfikacją. Po drugie, możemy również, w szerszym ujęciu, uruchomić program, by stwierdzić, czy system (ang. *superstructure*) sprzętowo-programowy, na którym program jest wykonywany, funkcjonuje właściwie. Oczywiście najczęściej testuje się programy w znaczeniu pierwszym, a tylko nieliczne, na przykład wyspecjalizowanie programy diagnostyczne, uruchamiane są w celu przetestowania systemów.

Oba rodzaje testowania są ściśle ze sobą związane. Program powinien być najpierw przetestowany celem sprawdzenia jego zgodności ze specyfikacją, ponieważ trudno jest badać poprawne działania systemu komputerowego, gdy nie wiemy, jak powinien zachowywać się program. Niepowodzenie testowania w znaczeniu pierwszym może świadczyć zarówno o błędzie programu, jak i o niewłaściwym funkcjonowaniu systemu, na którym program jest wykonywany. Zatem takie testowanie – w pewnym sensie – zakłada też właściwe funkcjonowanie systemu. Co więcej, najczęściej przyjmuje się, że pozytywny rezultat testu w znaczeniu pierwszym pociąga za sobą właściwe funkcjonowanie systemu. Należy jednak pamiętać, że błędny program również może dać dobre wyniki – na przykład na skutek nieprawidłowości w systemie (chociaż jest to bardzo mało prawdopodobne)[24]. Nie są to jednak jedyne znaczenia terminu „testowanie programu".

Jeśli tworzy się program komputerowy w celu rozwiązania jakiegoś problemu, to główną obawą staje się to, czy program spełni stawiane przed nim wymagania. Takie rozumienie programowania Colburn nazywa „inżynierią

---

[23] Podobnie jak obserwacja setek czarnych kruków potwierdza prawdziwość zdania: „Wszystkie kruki są czarne", choć go nie dowodzi.

[24] Kusząca jest perspektywa wyeliminowania tych złożonych zależności pomiędzy rodzajami testowania poprzez zastąpienie testowania w sensie pierwszym formalną weryfikacją programu, do której powrócimy w następnym paragrafie.

rozwiązań" (ang. *solution engineering*) z wykorzystaniem komputerów. Kluczową rolę odgrywa wtedy sposób rozwiązania problemu – czyli algorytm, oraz jego związek z programem.

Rolę algorytmu w komputerowym rozwiązywaniu problemów można wyjaśnić poprzez analogię z metodologią nauk przyrodniczych, szukających wyjaśnienia zjawisk obserwowalnych poprzez konstruowanie hipotez i ich eksperymentalne testowanie. Celem testowania hipotez jest ich obalenie lub potwierdzenie za pomocą interpretacji wyników eksperymentów. Podobnie możemy rozumieć rolę algorytmu w programowaniu. Formułuje się bowiem algorytm po to, by rozwiązać zadanie stawiane przed programem (podobnie jak formułuje się hipotezy w celu wyjaśnienia zjawiska). Następnie testuje się go specjalnie napisanym i uruchomionym programem (program realizuje algorytm). W zależności od wyników tego testu następuje akceptacja lub odrzucenie algorytmu (podobnie jak akceptacja lub odrzucenie hipotezy w naukach przyrodniczych). Zatem do przedstawionych powyżej celów testowania możemy dodać trzeci: uruchamianie programu dla sprawdzenia, czy dany algorytm rozwiązuje problem.

Jeśli przyjąć powyższą analogię pomiędzy testowaniem a eksperymentowaniem, to informatycy sprawdzają nie same programy, ale metody rozwiązywania problemów lub algorytmy, dla których programy te zostały zaprojektowane. Wtedy można przyjąć, że pisanie programów jest w informatyce tym samym, co konstruowanie modeli w naukach przyrodniczych, a testowanie programów jest niezbędne i nie można go w żaden sposób zastąpić.

Oczywiście traktowanie informatyki jako nauki eksperymentalnej ma również swoich przeciwników. Wskazują oni, między innymi, na problemy z określeniem różnic pomiędzy programem a algorytmem. Z jednej strony historia języków programowania pokazuje, że stają się one językami coraz wyższego poziomu, jednakże trzeba zauważyć, że języki do opisu algorytmów są coraz bardziej formalne. Zatem różnica pomiędzy algorytmem a programem powoli ulega zatarciu. Dowodem na to są na przykład prace zmierzające do automatycznego generowania programów na podstawie formalnej specyfikacji. Sukces w tej dziedzinie nie tylko całkowicie zlikwiduje różnice pomiędzy algorytmem a programem, ale również wyeliminuje „ludzkich" programistów. Warto zauważyć, że ogólną tendencją – nie tylko w programowaniu, ale również w weryfikacji – jest automatyzacja procesu przechodzenia od specyfikacji do programu. Konieczne jest wówczas przyjęcie dodatkowego założenia, że problem stawiany przed programem można sformułować na tyle jasno, by móc stworzyć jego formalną specyfikację.

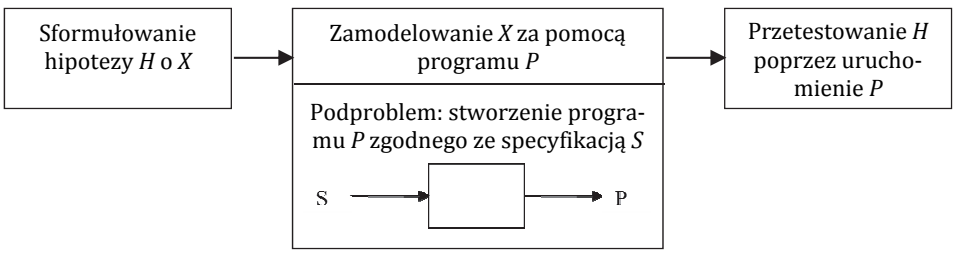
Niezależnie od formy formalizmu, jakiego użyto do zapisania tejże specyfikacji, konieczne jest sprawdzenie, czy zaproponowane w niej rozwiązanie

problemu jest poprawne. A zatem pojawia się kolejne zastosowanie dla testowania: uruchomienie programu w celu sprawdzenia, czy jego specyfikacja rozwiązuje problem. Metodologię programowania można wówczas przedstawić w następujący sposób:

1. Sformułowanie PROJEKTU ROZWIĄZANIA w terminach specyfikacji programu rozwiązującego ten problem.
2. TESTOWANIE projektu rozwiązania przez wygenerowanie i uruchomienie PROGRAMU.
3. AKCEPTACJA lub ODRZUCENIE projektu rozwiązania przez analizę rezultatów wykonania programu (Colburn, 2000, s. 170).

Takie sformułowanie pokazuje, podobnie jak w przypadku testowania algorytmów poprzez uruchomienie programów je realizujących, bliski związek metodologii programowania z metodologią nauk eksperymentalnych. Jest on równie dobrze widoczny zarówno w testowaniu programów, w którym uwzględnić należy pewne dodatkowe ograniczenia (na przykład dotyczące zasobów komputera używanego do rozwiązania problemu), jak i w procesie komputerowego sprawdzania hipotez naukowych.

Przypuśćmy, że pewien naukowiec – na przykład genetyk – próbując zrozumieć jakieś zjawisko lub proces, stworzył kilka jego modeli i zakodował je w postaci programów. I wtedy, uruchamiając program, testuje on konkretny model zjawiska – a nie procedury konieczne do jego zaimplementowania (program). W tym przypadku tworzenie programu i jego uruchomienie ma na celu nie tyle rozwiązanie problemu, ile sprawdzenie hipotezy. Istnieje zatem jeszcze jeden, piąty rodzaj testowania: uruchomienie programu dla sprawdzenia pewnej hipotezy w komputerowym modelu rzeczywistości. Oczywiście problem weryfikacji tego, czy program spełnia swoją specyfikację, jest w tym ujęciu jedynie podproblemem szerszego zagadnienia – a mianowicie wyjaśnienia zjawiska. Testowanie hipotez *H* na temat zjawiska *X* za pomocą programu *P* Colburn przedstawia schematycznie w następujący sposób:



Rys. 2. Testowanie hipotez za pomocą programu (Colburn, 2000, s. 173)

Eksperyment tworzenia i uruchamiania programów jest wówczas kluczowym elementem metodologii informatyki, a jedyną różnicą pomiędzy nim a eksperymentowaniem w naukach empirycznych jest to, że odbywa się on w komputerowym modelu rzeczywistości – w programie. Programowanie jest zatem nauką przyrodniczą, podobnie jak astronomia czy geologia[25]. Pogląd taki nazywany jest *paradygmatem naukowym*. Jego zwolennicy argumentują, że w informatyce, podobnie jak w innych naukach przyrodniczych, kluczową rolę odgrywa formułowanie i testowanie hipotez. Obiekty takie jak algorytmy czy specyfikacje programów można traktować jak hipotezy, które sprawdza się poprzez uruchomienie programów. Takie testowanie pokazuje adekwatność modelu świata rzeczywistego, jaki został wykorzystany w programie. Programy są wówczas tylko narzędziami w procesie sprawdzania, tak jak aparatura badawcza, i służą do potwierdzania lub obalania hipotez. Co więcej, istnieją również takie programy komputerowe, które są nie tylko narzędziami do testowania istniejących hipotez, ale służą również odkrywaniu nieznanych dotąd praw natury. Eden (2007) pisze: „[…] w bioinformatyce algorytmy genetyczne używane są do testowania stopnia, w jakim prawa selekcji naturalnej Darwina potwierdzają model reprodukcji molekuł, a w astronomii – do przewidywania modeli stworzenia wszechświata, które można testować za pomocą symulacji komputerowych" (s. 155).

Uprawianie informatyki jako nauki eksperymentalnej znajduje odzwierciedlenie w wielu aplikacjach zawierających symulacje. Dlatego właśnie dziedziną, w której takie jej stosowanie wydaje się najbardziej obiecujące, jest sztuczna inteligencja, gdzie ludzkie możliwości nie mają być wspierane przez programy, lecz symulowane za ich pomocą. W przypadku programów z dziedziny sztucznej inteligencji nie można stwierdzić nic na temat poprawności zawartego w nich modelu bez uprzedniego ich uruchomienia. Uruchamia się je bowiem po to, by przetestować hipotezę dotyczącą sposobu rozwiązywania problemów przez człowieka, a nie po to, by sprawdzić, czy program zgadza się ze specyfikacją. Dlatego też często badania w dziedzinie sztucznej inteligencji charakteryzuje się jako badania ludzkiej inteligencji – poprzez próby jej modelowania w programach.

O bliskim związku testowania programów z naukami eksperymentalnymi świadczyć może analiza używanych w nich rozumowań, przedstawiona przez Colburna (2000). Twierdzi on mianowicie, że argumentacje zawarte w testowaniu programów to rodzaj rozumowań indukcyjnych, ponieważ

---

[25] Już Newel i Simon (1976), pionierzy sztucznej inteligencji, twierdzili, że informatyka jako nauka o zjawiskach związanych z komputerami jest dyscypliną empiryczną i eksperymentalną.

wyciąga się w nich wnioski z obserwacji (wykonania programu dla zbioru danych wejściowych, które są sprawdzane) na temat tego, co nieobserwowalne (na temat działania programu dla wszystkich danych, również tych, których się nie testuje). Na przykład testując program obliczający sumę dwóch liczb rzeczywistych, sprawdza się jego działanie dla kilku przypadków i – jeśli wyniki są poprawne – wnioskuje się o poprawności tego programu dla wszystkich możliwych wejść. Jako takie, rozumowanie to nie jest mniej wiarygodne niż inne, wykorzystywane przez nauki eksperymentalne do wnioskowania na temat ogółu z obserwowalnych przypadków. Zależy ono w pewnym sensie od doboru badanej „próby" – to znaczy od tego, czy przypadki sprawdzane są typowe dla przypadków pomijanych (niebadanych). Jeśli bowiem przypadki nieobserwowane różnią się znacznie od przypadków badanych, to nie można stwierdzić, czy dla wszystkich danych wejściowych program daje poprawne rezultaty.

Możliwość obalenia jest charakterystyczna dla rozumowań empirycznych. Są to tak zwane rozumowania podważalne (ang. *defeasible*), które były badane zarówno przez filozofów (między innymi przez Pollocka), jak i przez kognitywistów. Są one zazwyczaj przeciwstawiane rozumowaniom niepodważalnym – choć rozróżnienie takie nie zawsze jest oczywiste. Rozważmy, za Colburnem, syntaktycznie podobne wnioskowania:

(A) 1. Ptaki są kręgowcami.
2. Fred jest ptakiem.
3. Zatem Fred jest kręgowcem.

(B) 1. Ptaki mogą latać.
2. Fred jest ptakiem.
3. Fred może latać.

Wnioskowania (A) i (B) wykorzystują inną relację konsekwencji. W rozumowaniu (A) mamy do czynienia z rozumowaniem niepodważalnym, gdyż oparte jest ono na niezawodnym schemacie wnioskowania (zdanie 1. poprzedzone jest dużym kwantyfikatorem). W przypadku rozumowania (B) nie można przedstawić argumentów w sposób adekwatny w języku rachunku predykatów, ponieważ kwantyfikatory nie są odpowiednie do zapisania zdania, że tylko typowe (nie wszystkie) ptaki latają. Co więcej, wykorzystana w tym przykładzie relacja konsekwencji nie zachowuje prawdziwości, tak jak odbywa się to we wnioskowaniu (A), lecz jedynie uzasadnienie. Oczywiście przesłanki rozumowania (B) można poszerzyć:

1. Typowe ptaki mogą latać.
2. Fred jest ptakiem.
3. Nie ma powodu, by uważać, że Fred nie jest typowy, to znaczy – Fred może latać.

Ale nawet jeżeli te przesłanki są prawdziwe, to nie mogą one gwarantować, że Fred może latać. Fred może mieć na przykład złamane skrzydło. Zatem prawdziwość wniosku 4. „Fred może latać" nie jest zagwarantowana dedukcyjnie: przesłanki mogą być prawdziwe, a wniosek jednak – fałszywy. Fakt, że wnioskuje się o własności, której nie mają wszystkie ptaki, powoduje, że wnioskowanie jest podważalne.

A co różni wnioskowania (A) i (B) z czysto logicznego punktu widzenia? Można uznać, że w drugim wnioskowaniu ukryty jest dodatkowy warunek – związany z jedną z przesłanek, warunek dotyczący „typowości" rozważanych przykładów. Takie ukryte założenia (przesłanki) można uważać za przykłady ogólnych zasad epistemicznych, bez których nie można przyswajać wiedzy. Zatem pełne tłumaczenie rozumowania zawartego w (B) powinno wyglądać następująco:

(C) 1. Typowe ptaki mogą latać.
2. Fred jest ptakiem.
3. Nie ma powodu, by przypuszczać, że Fred nie jest typowy, to znaczy, że Fred nie może latać.
4. Jeśli typowe ptaki mogą latać i Fred jest ptakiem, i nie ma powodu, by przypuszczać, że Fred nie jest typowy.
5. Zatem „Fred może latać" zostało uzasadnione[26].

Colburn twierdzi, że rozumowania leżące u podstaw testowania programów są również podważalne, ponieważ odnoszą się do założenia o „typowości" nieobserwowalnych przypadków. Na przykład w przypadku testowania programu obliczającego silnię, po sprawdzeniu kilku przykładów danych wejściowych (powiedzmy liczb od 0 do 5), zakładamy, że dalsze przypadki (to znaczy 6, 7, 8, …) są typowe – to znaczy, że nie różnią się w sposób zasadniczy od sprawdzonych. Oczywiście w bardziej zaawansowanych programach dobiera się testowane zbiory danych tak, by sprawdzić na przykład wszystkie możliwe ścieżki działania programu wynikające z jego schematu blokowego. Przetestowanie danych reprezentujących wszystkie możliwe przejścia programu redukują niebezpieczeństwo pozostawienia nieprzetestowanych tych przypadków, dla których program mógłby nie działać poprawnie. Jednakże dla bardzo dużych programów dobranie takich „reprezentatywnych" danych może być bardzo trudne. Rozumowanie dotyczące nieprzetestowanych przypadków jest podobne do rozumowania dotyczącego Freda – niesie ze sobą gwarancję uzasadnienia, a nie prawdziwości. Colburn (2000) opisuje je następująco:

---

[26] Warto w tym miejscu zauważyć, że przesłanka 4. powoduje, że wnioskowanie to nie jest dedukcyjne, ponieważ jest ona wyrażeniem modalnym w meta-języku o uzasadnieniu innego wyrażenia w języku przedmiotowym.

1. Program P jest poprawny dla zbadanych danych początkowych.
2. Nie ma powodu przypuszczać, że niezbadane dane nie są typowe (to znaczy, że sprawdzają P inaczej niż dane zbadane).
3. Zatem zdanie 'program P jest poprawny dla niezbadanych danych wejściowych' jest potwierdzone (s. 160).

Podważalna natura wnioskowań zawartych w testowaniu programów nie jest problemem dla sprawdzania ich wiarygodności. Jeśli jednak przyjmiemy, że celem sprawdzania tejże jest zagwarantowanie ich właściwego działania w sposób absolutny, to testowanie powinno zostać zastąpione – lub przynajmniej uzupełnione – formalnym dowodem.

2.2.3. Dowodzenie poprawności programów

Zwolennicy paradygmatu matematycznego uważają, że na wszystkich etapach tworzenia programów, od specyfikacji aż po weryfikację, ważną rolę odgrywają metody matematyczne. Już na pierwszym etapie, jakim jest tworzenie specyfikacji programu, konieczne staje się użycie języków i metod matematyki, ponieważ tylko specyfikacja formalna daje możliwość przeprowadzenia dowodu poprawności programu. Również implementacja programu komputerowego jest działalnością matematyczną. Użyteczność metod formalnych widać również w przypadku kompilatorów tłumaczących kod źródłowy programu na język maszynowy. Matematyka wykorzystywana jest tutaj dwojako: metod matematycznych używa się podczas samego procesu tłumaczenia – w celu uzyskania bardziej niezawodnego kodu maszynowego oraz udowodnienia, że program w języku maszynowym działa tak samo jak program źródłowy. Różnica w obu przypadkach polega na stosowaniu matematyki jako narzędzia inżynieryjnego oraz na używaniu jej w celu formalnego udowodnienia pewnych własności programów. Drugie z tych zastosowań wiąże się z jednym z najczęściej dyskutowanych w literaturze zagadnień filozofii informatyki, mianowicie z formalną weryfikacją programów i urządzeń, która posiada wszystkie cechy rozumowania matematycznego. Colburn (2000) pisze:

> Jednakże to, co jest matematyczne, to rozumowanie dotyczące programów w sensie abstrakcyjnym. Na przykład mając dany program P składający się ze stwierdzeń S_1, …, S_n i interpretując każde S_i jako stwierdzenia dotyczące maszyny abstrakcyjnej s_i, możliwe jest konstruowanie wyrażeń postaci:
>
> (T) Niech s_1, s_2, …, s_n będzie abstrakcyjną reprezentacją programu P.
> Wtedy P ma własność R,

gdzie R opisuje pewne aspekty wykonania programu P w sensie abstrakcyjnym. Na przykład R może opisywać ograniczenia czasu, jaki zajmuje wykonanie P, lub ilość pamięci, której wymaga wykonanie P. Przy dokładnej interpretacji s_i we właściwym języku oraz przy odpowiednim wyborze R, możliwe jest, by T było twierdzeniem w tym języku i zdaniem analitycznym (s. 136).

Zwolennicy paradygmatu matematycznego przez poprawność programu rozumieją zazwyczaj jego zgodność ze specyfikacją i twierdzą, że jedyną dopuszczalną metodą jej badania jest przeprowadzanie dowodów formalnych. Odrzucają przy tym wszelki eksperyment, w tym testowanie, jako metodę dochodzenia do wiedzy na temat programów. Wtedy wiedza informatyczna jest wiedzą *a priori*, dochodzimy do niej bowiem na drodze czystego rozumowania[27]. Za motto zwolenników takiego traktowania informatyki można przyjąć słowa C.A.R. Hoare'a (1969): „Programowanie komputerów jest nauką ścisłą, w której wszystkie własności programów i wszystkie konsekwencje ich wykonania w danym środowisku mogą, w zasadzie, być odkryte z samego tekstu programu za pomocą czysto dedukcyjnego rozumowania" (s. 576).

Badania nad metodami formalnego dowodzenia własności programów, w tym ich poprawności, prowadzone są od lat 60. XX wieku przez wielu znanych matematyków i informatyków. W wyniku tych prac powstały liczne techniki i narzędzia do opisu i badania własności programów – niektóre z nich wykorzystywane są do dzisiaj.

Jednym z pierwszych badaczy, którzy zajęli się perspektywami stworzenia informatyki matematycznej, był John McCarthy, który w artykule z roku 1962 przedstawił teorię matematyczną służącą do opisu i badania własności programów. Jego celem było stworzenie języka do wyrażania abstrakcyjnie pojmowanego obliczania. Zazwyczaj mówi się o obliczeniach w terminach funkcji[28], tak więc uznał on, że najlepszą matematyczną teorią bazową będzie teoria funkcji rekurencyjnych. Nie pozwala ona jednak na rozważania dotyczące instrukcji warunkowych, tej nieodłącznej części języków programowania – McCarthy wzbogacił zatem klasyczną teorię funkcji rekurencyjnych o wyrażenia warunkowe, to znaczy wyrażenia postaci

$$\texttt{if } p \texttt{ then } a \texttt{ else } b,$$

[27] Odpowiada to racjonalizmowi w epistemologii, który postuluje dochodzenie do wiedzy pewnej metodą rozumową. Dlatego też paradygmat matematyczny w informatyce jest czasami nazywany paradygmatem racjonalnym.

[28] McCarthy jest również autorem języka programowania Lisp, którego najważniejszym elementem jest funkcja.

gdzie *p* jest warunkiem logicznym. Wartością takiego wyrażenia warunkowego jest *a*, jeśli warunek *p* jest spełniony (*p* ma wartość *true*), albo *b* – w przeciwnym przypadku (gdy *p* przyjmuje wartość *false*)[29].

Celem McCarthy'ego było stworzenie dostatecznie silnego aparatu matematycznego, aby móc dowodzić, że programy są poprawne w rozumieniu szerszym niż tylko ich zgodność ze specyfikacją: „chcielibyśmy móc dowieść, że dana procedura rozwiązuje dany problem" (McCarthy, 1962, s. 3). Co prawda, teorii takiej nie udało mu się stworzyć, ale powstało wiele narzędzi użytecznych w dowodzeniu pewnych relacji zachodzących pomiędzy algorytmami. McCarthy przedstawił m.in. listę aksjomatów definiującą wspomnianą już operację warunkową:

```
1)(if true then a else b)=a
2)(if false then a else b)=b
3)if p then (if q then a else b) else (if q then c
  else d)= if q then (if p then a else c) else (if
  p then b else d)
4)f(if p then a else b) = if p then f(a) else f(b)
```

oraz opisał metodę stosowania tak rozumianej instrukcji do wyrażania niektórych cech programowania w języku funkcji rekurencyjnych. Rozszerzona przez McCarthy'ego teoria funkcji rekurencyjnych miała również umożliwić automatyczne tłumaczenie pomiędzy językami programowania[30]. Twierdził on, że debugowanie (usuwanie usterek w programach) da się wyeliminować poprzez podanie dowodu, że program zgadza się ze specyfikacją – który to fakt może być sprawdzony automatycznie przez program komputerowy. Wielu autorów wypowiadało się później na temat tego, co oznacza „zgodność programu ze specyfikacją" oraz – jak dowód takiej zgodności powinien wyglądać.

Najczęściej twierdzi się, że poprawność programu (jego zgodność ze specyfikacją) to relacja zachodząca pomiędzy wejściem a wyjściem programu, spełniająca warunki nałożone przez tę specyfikację (Turner, 2011). Dokładniej – jeśli *p* jest programem, to spełnia on specyfikację *R*, która jest relacją pomiędzy wejściem typu *I* oraz wyjściem typu *O*, gdy zachodzi następujący warunek:

dla wszystkich wejść *i* typu *I*, para (*i*, *p*(*i*)), spełnia relację *R*,

gdzie *p*(*i*) jest rezultatem wykonania programu *p* na wejściu *i*. *R* wyrażona jest przy tym w pewnym języku specyfikacji, natomiast *p* – w języku

---

[29] Wiele funkcji w matematyce można zdefiniować za pomocą wyrażeń warunkowych, np. (1) wartość bezwzględna: |x|=if x<0 then –x else x, (2) silnia: n!=g(n,1), gdzie *g*(*n*,*s*)=if *n*=0 then *s* else *g*(*n*-1,*n***s*).

[30] Częściową realizacją pomysłu takiego tłumaczenia są kompilatory.

programowania. Język specyfikacji jest zazwyczaj pewnym wariantem logiki predykatów, a sam dowód poprawności programu przeprowadzany jest w odpowiednim systemie logicznym. Jedną z pierwszych logik służących do dowodzenia poprawności programów stworzył w roku 1969 Hoare.

Podane przez Hoare'a aksjomaty i reguły w zamierzeniu miały opisywać zachowanie programów. Hoare uważał, że poprawne działanie programu można wyrazić w terminach związków zachodzących pomiędzy wartościami pewnych zmiennych przed jego wykonaniem i po nim[31] (warunki te można sformułować w terminach logiki i odpowiednich operacji arytmetycznych). Wprowadził symbol $P\{Q\}R$ oznaczający: „jeśli warunek P jest prawdziwy przed wykonaniem programu Q, to warunek R będzie prawdziwy po jego zakończeniu". Używając takich oznaczeń, sformułował aksjomaty i reguły dowodowe systemu mającego na celu dowodzenie poprawności programów.

D0. Aksjomat podstawiania[32]:

$$\vdash P_0\{x := f\}P\,,$$

gdzie: x jest identyfikatorem zmiennej, f jest wyrażeniem języka programowania bez efektów ubocznych, którego wartość jest podstawiana do zmiennej (może ono zawierać zmienną x), P_0 jest wynikiem podstawienia f za wszystkie wystąpienia x do P[33].

D1. Reguły wynikania:

$$\text{Jeśli } \vdash P\{Q\}R \text{ oraz } \vdash R \supset S, \text{ to } \vdash P\{Q\}S.$$

$$\text{Jeśli } \vdash P\{Q\}R \text{ oraz } \vdash S \supset P, \text{ to } \vdash S\{Q\}R.$$

Pierwsza z reguł stwierdza, że jeśli wykonanie programu Q zapewnia prawdziwość warunku R, to zapewnia również prawdziwość każdego warunku, który wynika z R. Druga natomiast mówi, że jeśli P jest warunkiem prawdziwym przed wykonaniem programu Q, po którego wykonaniu prawdziwy jest warunek R, to przed wykonaniem programu Q prawdziwe jest również każde założenie, które pociąga za sobą warunek P.

D2. Reguła składania:

$$\text{Jeśli } \vdash P\{Q_1\}\ R_1 \text{ oraz } \vdash R_1\ \{Q_2\}\ R, \text{to } \vdash P\{(Q_1;Q_2)\}R,$$

---

[31] Hoare przyjął tutaj notację wprowadzoną przez Floyda w pracy z roku 1967, ale zastosował ją w sposób bardziej ogólny.

[32] Jak twierdzi Hoare, podstawianie jest jedną z najbardziej charakterystycznych operacji w programowaniu oraz taką, która odróżnia informatykę od innych gałęzi matematyki.

[33] D0 nie jest pojedynczym aksjomatem, ale schematem aksjomatów.

gdzie $(Q_1;Q_2)$ oznacza złożenie (następujące po sobie wykonanie) programów Q_1 i Q_2.

Hoare rozpatruje również operację iteracji, niezbędną w językach programowania. Może być ona realizowana za pomocą pętli while *B* do *S*, gdzie *B* jest warunkiem logicznym, a *S* jest fragmentem programu. Podczas wykonania programu, jeśli *B* ma wartość logiczną *true*, to *S* jest wykonywane; w przeciwnym przypadku – *S* jest pomijane. Zatem *S* wykonywane jest tak długo, dopóki *B* nie przyjmie wartości *false*. Oczywiście zawsze po zakończeniu pętli prawdziwy jest warunek $\neg B$, co pokazuje następująca reguła:

D3. Reguła iteracji:

$$\text{Jeśli } \vdash P \wedge B\{S\}P \text{, to } \vdash P\{\text{while } B \text{ do } S\} \neg B \wedge P.$$

W tak zdefiniowanym systemie Hoare dowodzi, w sposób czysto formalny, poprawności programu obliczającego iloczyn i resztę z dzielenia dwóch liczb całkowitych. Wydaje się zatem, że odniósł sukces; jednak miał on świadomość pewnych ograniczeń związanych z takim sposobem dowodzenia poprawności programów. Po pierwsze, operacja podstawiania nie może powodować efektów ubocznych, czyli metody tej nie można zastosować do wszystkich języków programowania. Po drugie, poprawność działającego programu zależy od czegoś więcej niż tylko od weryfikacji formalnej programu, między innymi od sposobu reprezentacji liczb, ograniczeń czasowych związanych z systemem operacyjnym i od metod radzenia sobie z problemem przepełnienia. Po trzecie, w przytoczonym systemie nie jest możliwe przeprowadzanie dowodu z tego, że program zakończy swoje działanie. Dlatego też dla programów, które mają dowody poprawności w przedstawionym powyżej systemie, Hoare proponuje używanie terminu „program częściowo poprawny" – w odróżnieniu od programów „poprawnych", to znaczy takich, które zatrzymują się i są częściowo poprawne[34].

Próbując wyeliminować ograniczenie drugie, Hoare podał aksjomatykę dla „arytmetyki komputerowej", zwracając przy tym uwagę na to, że niektóre własności arytmetyki liczb całkowitych mogą być inne niż arytmetyki komputerowej. Na przykład w zwykłej arytmetyce prawdziwe jest twierdzenie, że $\sim\exists x\forall y(y \leq x)$ – to znaczy nie istnieje liczba największa, podczas gdy w każdej arytmetyce skończonej (a taką właśnie jest arytmetyka komputerowa) prawdziwe jest zdanie $\forall x(x \leq max)$, gdzie max oznacza największą reprezentowalną liczbę całkowitą. Przedstawił on również trzy aksjomaty opisujące sytuację przepełnienia, to znaczy przypadek, gdy żąda

[34] Widać tu wyraźną analogię pomiędzy pojęciami częściowej i całkowitej poprawności algorytmów.

się umieszczenia w pamięci komputera liczby większej niż max, w zależności od sposobu, w jaki traktujemy to przepełnienie: a) operacje powodujące przepełnienie są niewykonalne: $\sim\exists x(x = max + 1)$, b) wynikiem operacji powodującej przepełnienie jest największa reprezentowalna liczba: *max* = *max* + 1, c) wynik takiej operacji obliczany jest modulo rozmiar zbioru reprezentowalnych liczb całkowitych: *max* + 1 = 0 (mod *max*).

Również inni badacze próbowali stworzyć matematyczne podstawy formalnych badań własności programów, a niektóre z ich osiągnięć wykorzystywane są także i dzisiaj. Tak jest na przykład w przypadku *metody niezmienników Floyda*, zapoczątkowanej przez badania Petera Naura.

Naur (1966) uważał, że jednym ze sposobów dowodzenia poprawności programów jest traktowanie ich równocześnie jako obiektów statycznych (napisany tekst), jak i dynamicznych (wykonywalnych). Sposobem na mówienie o programach w tak, zdawałoby się, różnych kategoriach jest zatrzymanie wykonania programu od czasu do czasu i opisanie stanu maszyny w momencie zatrzymania. Punkty zatrzymania programu Naur nazywał „zdjęciami" (ang. *snapshots*). Jeśli w programie umieści się odpowiednią liczbę punktów kontrolnych (zdjęć), będzie można wówczas badać związki, jakie – wyrażone za pomocą formuł – zachodzą w nich pomiędzy stanami maszyny. Korzystając z formuł w poprzednich punktach kontrolnych, można przeprowadzić dowód dotyczący stanu programu w kolejnym punkcie. Naur używał stworzonej przez siebie metody nie tylko do dowodzenia poprawności programów, lecz również jako narzędzia pomocniczego przy bieżącej konstrukcji elementów programu – a szczególnie pętli. Wierzył, że komponenty matematyczne w programowaniu są niezbędne. Dał on również początek – wraz z Robertem W. Floydem – współczesnym badaniom własności programów z użyciem semantyki.

W 1967 roku Floyd przedstawił pomysł przypisania znaczenia programom w terminach formuł umieszczonych na diagramach sekwencji działań tych programów (a właściwie algorytmów). Jest to jedna z metod dowodzenia częściowej poprawności[35] algorytmów, nazywana współcześnie metodą niezmienników Floyda[36].

W metodzie tej wykorzystuje się przedstawienie algorytmu w postaci diagramu sekwencji działań, traktowanego jako graf skierowany, w którego wierzchołkach znajdują się operacje. Interpretacja *I* takiego diagramu to

---

[35] Do całkowitej poprawności należy udowodnić jeszcze, że algorytm zatrzymuje się dla wszystkich danych.

[36] Ideę Floyda rozwinął później C.A.R. Hoare pod nazwą „metoda niezmienników", dlatego też współcześnie nazywa się ją *metodą niezmienników Floyda*. Jest to jedna z najlepiej opracowanych matematycznie technik dowodzenia poprawności programów.

przyporządkowanie jego krawędziom pewnych warunków logicznych, przy czym niektóre ze zmiennych wolnych występujących w tych warunkach mogą być zmiennymi, na których działa program. Dla każdej krawędzi *e* warunek *I*(*e*) nazywa się znacznikiem (ang. *tag*). Z każdą operacją *c* (wierzchołkiem grafu) mogą być skojarzone wejścia $a_1, a_2, ..., a_k$ (krawędzie „wchodzące" do bloku) i wyjścia $b_1, b_2, ..., b_l$ (krawędzie „wychodzące" z bloku). Przez $P_i (1 \le i \le k)$ oznacza się znaczniki związane z wejściami, natomiast przez $Q_i (1 \le i \le l)$ – znaczniki związane z wyjściami. Weryfikacja opiera się na udowodnieniu dla każdej operacji *c* diagramu, że jeśli znacznik P_i związany z jakąkolwiek strzałką a_i prowadzącą do bloku jest prawdziwy, zanim zostanie wykonana operacja *c* opisywana przez blok, to wszystkie znaczniki Q_j związane ze strzałkami wychodzącymi z bloku b_j są prawdziwe po wykonaniu tej operacji. Wtedy przez indukcję względem liczby wykonanych operacji można pokazać pewne związki pomiędzy wartościami początkowymi a wartościami końcowymi. Rozważmy następujący przykład:

```
silnia (N)
begin
        M:=N
        while N>1 do
        begin
              N:=N-1
              M:=M*N
        end
return M
end
```

W tym przypadku warunek początkowy to: „N jest liczbą naturalną", natomiast warunek końcowy, który należy udowodnić, to: M=N!.

Aby pokazać poprawność warunku końcowego, należy udowodnić niezmiennik pętli, z którego on wynika. Niezmienniki takie zazwyczaj stwierdzają coś o zmiennych po określonej liczbie iteracji tej pętli.

Niech N_i oraz M_i oznaczają zawartości zmiennych *M* i *N* po *i* przejściach pętli. Niezmiennik, jaki należy udowodnić, ma postać: $M_i \cdot (N_i - 1)! = N!$. Wykorzystujemy zasadę indukcji matematycznej (indukcja po *i*).

1. gdy *i*=1, to $M_i = N$ oraz $N_i = N$, zatem

$$M_i \cdot (N_i - 1)! = N \cdot (N - 1)! = N!$$

2. krok indukcyjny
   założenie indukcyjne: $M_i \cdot (N_i - 1)! = N!$
   teza indukcyjna $M_{i+1} \cdot (N_{i+1} - 1)! = N!$

W programie widać, że po każdej iteracji pętli zmienna *M* zawiera to, co zawierała w poprzednim kroku, pomnożone przez wartość *N* po iteracji. Zatem $M_{i+1} = M_i \cdot N_{i+1}$. Ponadto $N_{i+1} = N_i - 1$. Wtedy:

$$M_{i+1} \cdot (N_{i+1} - 1)! = M_i \cdot (N_i - 1) \cdot (N_i - 2)! = M_i \cdot (N_i - 1)! = N!$$

Jest to typowy dowód matematyczny, niektórzy (np. Colburn, 2000) twierdzą jednak, że nie daje on stuprocentowej gwarancji co do poprawności wykonania programu.

Rozważmy podstawową przesłankę kroku indukcyjnego: „po każdej iteracji pętli zmienna *M* zawiera to, co zawierała w poprzednim kroku pomnożone przez wartość *N* po iteracji". Pamiętając o dwuznaczności Fetzera, związanej ze statusem ontologicznym *M* i *N*, warto zauważyć, że jeśli traktujemy je jako abstrakcyjne miejsca w pamięci abstrakcyjnej maszyny, to dowód jest rzeczywiście poprawny, gdyż maszyna taka jest idealna w tym sensie, że operacje na niej nigdy nie zawodzą – a więc przesłanka kroku indukcyjnego jest uzasadniona ponad wszelką wątpliwość. Jeśli jednak *M* i *N* potraktujemy jako fizyczne miejsca w pamięci maszyny fizycznej, to przesłankę tą można uzasadnić tylko w oparciu o dodatkowe założenia dotyczące, między innymi, właściwego funkcjonowania sprzętu. Wtedy założenia te nadają przeprowadzonej weryfikacji charakter „relatywny", a nie „absolutny". Zatem rozumowanie dotyczące zachowania programu na fizycznej maszynie, nawet na bazie dowodu indukcyjnego, jest podważalne. Trzeba bowiem poczynić pewne, być może fałszywe założenia dotyczące typowości nieobserwowalnych przypadków, by wywnioskować, że będą one dawały poprawny wynik. W przypadku formalnej weryfikacji programu są to założenia o typowości, czyli poprawnym funkcjonowaniu używanego oprogramowania i sprzętu.

Wykorzystywanie metod matematycznych w informatyce, pomimo krytyki zapoczątkowanej pod koniec lat 70. XX wieku[37], miało wielu zagorzałych zwolenników. Jednym z nich był E.W. Dijkstra, od którego pochodzą dwa z trzech najbardziej znanych argumentów na rzecz paradygmatu matematycznego w informatyce.

Pierwszy z nich nazwać można *argumentem ontologicznym*[38], ponieważ Dijkstra twierdzi, że programy jako obiekty abstrakcyjne mają pierwszeństwo ontologiczne przed maszynami (bytami fizycznymi). Program rozumiany jest wówczas jako ciąg operacji na symbolach abstrakcyjnych,

---

[37] Powszechnie uważa się, że praca DeMillo, Liptona i Perlisa z roku 1979 *Social Processes and Proofs of Theorems and Programs* zapoczątkowała debatę na temat możliwości i metod weryfikacji programów.

[38] Nazwę tę zaproponował Fetzer w pracy z 1991 roku.

a komputer jedynie wykonuje go i przedstawia wyniki. Język programowania, w którym program został zapisany, można traktować jako formalny system, dla którego program jest jedynie modelem, a o programach powinniśmy myśleć „nawet bez wspominania o ich możliwych 'zachowaniach'" (Dijkstra, 1989, s. 1403). Głównym zadaniem programisty, poza napisaniem programu, staje się wówczas stworzenie dowodu formalnego pokazującego, że program ten spełnia formalną specyfikację.

Ten sam autor podał drugi, wspomniany w poprzednim paragrafie argument na rzecz stosowania metod formalnych w informatyce, twierdząc, że testując program, nie można wykazać, że nie ma w nim błędów; można jedynie wykazać ich obecność. Stwierdzenie to można nazwać *argumentem epistemologicznym*. Jest ono zgodne z Popperowską koncepcją metodologii nauk, w której hipotezy naukowe mogą zostać obalone, ale nie mogą zostać potwierdzone. Sugeruje ono jednocześnie, że jedyną drogą do uzyskania wiedzy na temat poprawności programów jest zastosowanie metod formalnych.

Dijkstra twierdzi również, że informatyka jest i zawsze będzie połączeniem „obliczania" (wykonywanego przez komputery) oraz „programowania" (operacji na symbolach przeprowadzanych przez człowieka) – a „programowanie automatyczne" jest pojęciem wewnętrznie sprzecznym, ponieważ sugeruje możliwość eliminacji człowieka z procesu programowania. Zgadzając się z tym poglądem, można jednoznacznie stwierdzić, że w klasyfikacji wszystkich nauk miejsce informatyki znajduje się tuż obok matematyki i logiki formalnej[39].

Trzeci argument na rzecz paradygmatu matematycznego, nazywany zazwyczaj *argumentem ze zwiększania skali* (ang. *scaling-up argument*), związany jest z praktyczną realizacją formalnej weryfikacji programów. Weryfikacja taka jest zagadnieniem stosunkowo młodym, jednakże można już teraz dokonywać weryfikacji prostych programów – wnioskujemy stąd, że niebawem będzie można weryfikować coraz bardziej złożone algorytmy i coraz bardziej skomplikowane programy. Produkowane w rzeczywistości duże systemy nie są bowiem niczym więcej niż tylko złożeniem prostych algorytmów i modeli. Ponadto raz zweryfikowany algorytm czy model programu będzie można powiększać – aż do rozmiarów rzeczywistego systemu. A zatem weryfikacja dużego systemu będzie sumą wielu małych weryfikacji jego składowych, a z tego wynika, że z czasem powstaną formalne weryfikatory dla rzeczywistych, nawet bardzo skomplikowanych programów.

---

[39] Dijkstra proponuje nawet nazywanie informatyki skrótem VLSAL, od słów **V**ery **L**arge **S**cale **A**pplication of **L**ogic.

Powyższa argumentacja jest szeroko krytykowana. Na podstawie jej analizy DeMillo, Lipton i Perlis (1979) doszli do przeciwnego wniosku; stwierdzili, że weryfikatory (programy automatycznie sprawdzające inne programy) dla rzeczywistych systemów nigdy nie powstaną.

Najwięcej wątpliwości budzi jednak twierdzenie, że duży system jest niczym więcej niż tylko sumą swoich składowych. W praktyce programistycznej stosuje się wiele metod i trików w łączeniu tychże składowych, co w efekcie prowadzi do powstania zupełnie nowych obiektów, które mają pewne cechy dodatkowe. Co więcej, oszacowano, że około połowy kodu działających systemów to tak zwany *inferface* i obsługa komunikatów o błędach, które są tworzone *ad hoc* i jako takie nie są weryfikowalne.

Po drugie, nie powinno się utożsamiać algorytmów i programów, wystarczy bowiem porównać ich specyfikacje, aby stwierdzić, że są to inne obiekty. Specyfikacje algorytmów są zwięzłe i jasno określone, podczas gdy specyfikacje rzeczywistych systemów są ogromne, często tego samego poziomu komplikacji, co same systemy. Specyfikacje algorytmów są stabilne, czasami nie zmieniają się przez stulecia, natomiast specyfikacje rzeczywistych programów zmieniają się z dnia na dzień, a nawet z godziny na godzinę! Nie jest to zatem różnica w skali, ale w rodzaju. Rozwój jednych nigdy nie doprowadzi wprost do drugich. Obiekty te są zasadniczo różne.

Po trzecie, nie zawsze można przewidzieć działanie programu w sposób czysto analityczny, ponieważ tworzenie oprogramowania jest procesem nieciągłym. Zmiana linii – lub czasami tylko jednego bitu – w kodzie źródłowym może zupełnie zmienić działanie programu w sposób, którego nie rozumiemy i nie jesteśmy w stanie przewidzieć[40]. Wielu programistów zauważa, że zachowanie napisanych przez nich programów bywa zaskakujące, a nawet nieprzewidywalne[41]. Również wiele wirusów zmienia działanie programów, a w dobie Internetu prawie każdy komputer narażony jest na ich ataki i podlega ryzyku modyfikacji znajdującego się na nim oprogramowania. Co więcej, rzeczywiste systemy muszą być serwisowane i modyfikowane, a więc zmieniane[42]. Poczynione tu uwagi można nazwać *argumentem*

---

[40] W systemach chaotycznych nawet drobne zmiany danych początkowych mogą pociągać za sobą nieprzewidywalne zmiany wyników – efekt taki określa się często mianem „efektu motyla". Można go zaobserwować nie tylko w programowaniu, ale również w przypadku zachowań mikroprocesorów (por. Berry, Perez i Temam, 2006).

[41] Efekt zaskoczenia pracą programu podkreślali Appel i Haken (1983) – autorzy komputerowego dowodu twierdzenia o czterech barwach.

[42] Istnieją nawet propozycje nowego paradygmatu w programowaniu nastawionego na wprowadzanie zmian w programach. Vaclav Rajlich w pracy *Changing the Paradigm of Software Engineering* (Rajlich, 2006) twierdzi, że wprowadzanie zmian w istniejących systemach to obecnie jedna z podstawowych działalności programistów.

z nieprzewidywalności – przeciw weryfikacji programów. Oczywiście dostatecznie zmotywowany informatyk mógłby poświęcić wiele lat, by zweryfikować znaczący fragment programu, jednak zdobędzie się na to tylko wtedy, jeśli będzie przekonany, że program ten nie ulegnie zmianie.

Nawet przyjęcie założenia, że raz napisany program się nie zmienia, nie prowadzi jeszcze do wniosku, że formalna weryfikacja rzeczywistych programów jest w ogóle możliwa. Dowody formalne w matematyce, nawet dla bardzo prostych teorii, są często długie i trudne do zrozumienia. Podobnie jest w przypadku formalnych weryfikacji rzeczywistych programów. Co prawda, sugeruje się czasem, że języki programowania wysokiego poziomu ułatwiają czytanie programów i ich szybkie zrozumienie, ale wystarczy spróbować zrozumieć fragment niedbale napisanego programu, by stwierdzić, że nie jest to zawsze prawda. Nie ma również powodu, by przyjąć, że weryfikacja programu po ewentualnych modyfikacjach jest łatwiejsza lub krótsza niż weryfikacja programu wyjściowego, co podważa sens weryfikacji jako takiej. Co więcej, istnieją programy, na przykład kontrolujące ruch lotniczy, składające się z miliona linii kodu, które nie mogą zostać sprawdzone w sposób czysto matematyczny. Nie jest to niemożliwość logiczna, lecz praktyczna: złożoność systemu jest za duża. Można zatem sformułować *argument ze złożoności* przeciw formalnej weryfikacji jako stwierdzenie, że stworzenie weryfikacji funkcjonujących w rzeczywistości programów nie jest możliwe ze względu na ich złożoność.

DeMillo, Lipton i Perlis twierdzą ponadto, że weryfikacje programów, mimo że są ciągami formuł logicznych, nie są dowodami w sensie matematycznym. Akceptacja wyników w matematyce to pewien proces społeczny, niewystępujący w informatyce, prowadzący do przekonania o poprawności wyniku, w którym tylko jedną z części jest dowód i co więcej, bardzo rzadko jest to dowód formalny (rozumiany jako łańcuch następujących po sobie formuł logicznych). Ich zdaniem współczesna matematyka jest nieformalnym, intuicyjnym procesem socjologicznym zachodzącym wewnątrz pewnej wspólnoty. Poza tym nie ma możliwości opisania historii rozwoju pojęć i dyscyplin matematycznych bez szczegółowej analizy tych procesów, szczególnie tych, które dotyczą dowodzenia[43]. Matematyk popełnia błędy, ale ważne jest to, że błędy te są poprawiane, i to nie dzięki używaniu logiki formalnej, ale przez zaangażowanie innych matematyków znajdujących luki w dowodach, na przykład dzięki wzajemnemu powiązaniu twierdzeń matematycznych. Informatyk z kolei może stworzyć i opublikować własną

[43] DeMillo, Lipton i Perlis podają wiele przykładów świadczących o tym, że uzyskane wyniki mogą przeczyć sobie nawzajem – o czym piszą już sami ich autorzy w swoich publikacjach.

weryfikację, ale nie może zmusić nikogo do jej przeczytania – na przykład w sytuacji, gdy czytelnik będzie zdawał sobie sprawę, że nie ma ona i prawdopodobnie nie będzie mieć nic wspólnego z innymi weryfikacjami. Nie może więc zaistnieć proces społeczny, który opisują autorzy publikacji z 1979 roku. Przyjrzyjmy się teraz przebiegowi takiego procesu.

Początkowo dowód twierdzenia jest wiadomością, a nie istniejącym niezależnie obiektem abstrakcyjnym. W pierwszej odsłonie jest to wiadomość słowna lub naszkicowana na tablicy albo skrawku papieru. Taka właśnie wiadomość poddawana jest pierwszym ocenom podczas dyskusji w gronie matematyków, na seminariach lub w prywatnych rozmowach. Jeśli wynik nie wzbudza zaufania lub nawet zainteresowania wśród współpracowników, mądry matematyk rozważa go ponownie. Ale jeśli zostanie przyjęty, choćby z umiarkowanym zainteresowaniem i zaufaniem, autor spisuje dowód. Jego zapis krąży przez jakiś czas wśród zainteresowanych w formie brudnopisu. Jeśli po pewnym czasie nadal wydaje się on wiarygodny, wówczas autor zgłasza dopracowaną wersję dowodu do publikacji. Gdy recenzenci również uznają go za przekonujący, dowód jest publikowany i wtedy może być czytany przez szersze grono odbiorców. Z kolei jeśli czytający dowód matematyk uwierzy „na pierwszy rzut oka" w jego poprawność, to zachodzi u niego proces „internalizacji" tego wyniku. Podejmuje on próbę przeformułowania go w taki sposób, by wyrazić go w swoich terminach („własnymi słowami"), które pasują do jego osobistego poglądu na wiedzę matematyczną. Nie ma dwóch matematyków, u których internalizacja następuje dokładnie w ten sam sposób, więc w wyniku takiego procesu powstaje zazwyczaj wiele wersji tego samego twierdzenia – a każda z nich zwiększa przekonanie społeczności matematyków o tym, że oryginalne twierdzenie jest prawdziwe. Wiarygodne twierdzenia często znajdują zastosowanie, na przykład – jako lematy w większych twierdzeniach, przenikając do pokrewnych gałęzi matematyki. Jeśli twierdzenie lub technika dowodowa sprawdzają się w innej dziedzinie niż początkowa, to zwiększa się przekonanie co do poprawności wyjściowego wyniku. Po takiej internalizacji, transformacji, generalizacji i użyciu w różnych działach matematyki społeczność uznaje wynik za poprawny i o twierdzeniu mówi się, że jest prawdziwe w klasycznym sensie – czyli że poprawność ta może być pokazana formalnie, w sposób dedukcyjny. Jednak dla większości twierdzeń matematycznych nigdy nie przeprowadza się w pełni formalnych dowodów, ponieważ byłyby one zbyt długie, a ich istnienie jest zbędne. Matematykom do uznania wiarygodności dowodu wystarcza przekonanie o możliwości jego formalizacji.

DeMillo, Lipton i Perlis uważają, że istnieje pewna analogia pomiędzy matematyką a programowaniem – jednakże nie taka, jaką przyjmują zazwyczaj zwolennicy weryfikacji formalnej:

Tabela 4. Analogia pomiędzy matematyką a programowaniem 1

matematyka	*programowanie*
twierdzenie	program
dowód	weryfikacja

Źródło: DeMillo, Lipton i Perlis, 1979, s. 275

ale następująca:

Tabela 5. Analogia pomiędzy matematyką a programowaniem 1

matematyka	*programowanie*
twierdzenie	specyfikacja
dowód	program
teoretyczny dowód formalny	weryfikacja

Źródło: DeMillo, Lipton i Perlis, 1979, s. 275

Warto zauważyć, że większość zwolenników paradygmatu matematycznego rozumie matematykę jako teorię formalną, grę symboli, nie uwzględniając jej rzeczywistego społecznego charakteru. W rzeczywistości dowody formalne jedynie zwiększają przekonanie matematyków co do prawdziwości twierdzeń – a jest ono wynikiem opisanych powyżej mechanizmów społecznych. Brak tych mechanizmów skazuje na niepowodzenie tak zwane dowody poprawności programów, formalne weryfikacje, które nie odpowiadają przeprowadzanym w praktyce dowodom matematycznym. Weryfikacje nie są wiadomościami, nikt nie próbuje nawet zainteresować współpracowników wydrukiem weryfikacji, szczególnie przeprowadzonej w sposób zupełnie automatyczny, bo może zostać uznany za dziwaka. Co więcej, bardzo długie weryfikacje nie zawsze mogą zostać przeczytane, a jako takie nie mogą zostać zinternalizowane, przeformułowane, uogólnione lub włączone do innych dyscyplin. Nie zyskują one wiarygodności stopniowo, tak jak dowody matematyczne, „można w nie wierzyć, ślepo, jak w czysty akt wiary, albo wcale" (DeMillo, Lipton i Perlis, 1979, s. 275).

Jak już wspomniano, zwolennicy metod matematycznych w informatyce przyjmują najczęściej, że pojęcie poprawności programu sprowadza się do jego zgodności ze specyfikacją. Powstaje wtedy problem natury logicznej, a właściwie metodologicznej, dotyczący specyfikacji nieformalnych. Jeśli bowiem wymagania co do programu zostały wyrażone w sposób nieformalny, na przykład w języku naturalnym, a sam program ma charakter formalny – to w weryfikacji musi istnieć jakieś stadium przejściowe, które

powinno mieć charakter nieformalny. Nie bardzo wiadomo, w jaki sposób pogodzić istnienie takiego nieformalnego elementu – z ideą czysto formalnego dowodu. Zatem specyfikacja programu, który ma zostać automatycznie zweryfikowany, musi być specyfikacją formalną.

Twierdzi się często, że formalna specyfikacja programu nie tylko pozwala na automatyczną jego weryfikację, ale również zabezpiecza go przed błędami i niespójnością (Meyer, 1985). Peter Naur (1982) pokazuje prosty przykład, który obala to twierdzenie.

Rozważmy program, który oblicza początkowe liczby pierwsze aż do liczby *n* podanej przez użytkownika oraz wypisuje je w kolejności rosnącej (czyli od najmniejszej do największej). Jones w swojej książce (Jones, 1980) buduje specyfikację tego problemu, rozpoczynając od nieformalnego jej sformułowania, podobnego do powyższego, aż do jej postaci formalnej:

$$\text{let } n = \text{hd } il$$

$$\text{elems } ol = \{i \mid 1 \le i \le n \wedge is - prime(i)\} \wedge \text{is} - ordered(ol).$$

Tak sformułowana specyfikacja wskazuje listę „wejść" *il* i listę „wyjść" *ol* oraz stwierdza, że po wykonaniu programu, jeśli *n* oznacza pierwszy element listy wejściowej, na liście wynikowej znajdują się uporządkowane liczby pierwsze pomiędzy 1 a *n*[44]. Specyfikacja ta dokładnie określa, jakie liczby powinny znaleźć się na liście wynikowej, ale nie stwierdza, w jaki sposób mają zostać pokazane ani też – w jakim formacie.

Przypuśćmy, że na podstawie powyższej specyfikacji powstał program, który wypisuje na drukarce właściwe liczby w postaci szesnastkowej lub ósemkowej. Czy osoba korzystająca z tego programu uzna go za poprawny? W tym przypadku równie ważne jak określenie tego, co ma zostać obliczone, jest założenie dotyczące sposobu wyświetlania wyniku. Zatem wiarygodność programu w dużej mierze zależy od formy komunikowania się programu z człowiekiem, a tego zazwyczaj nie określają specyfikacje formalne.

Krytykując ideę specyfikacji formalnych, C. Floyd (1987) zwraca uwagę na zmienność jako na nieodłączną cechę procesu tworzenia specyfikacji. Rozróżnia on dwie perspektywy patrzenia na ten proces.

Pierwsza to perspektywa zorientowana na produkt (ang. *product-oriented perspective*). Zakłada się w niej, że problem, dla którego szukamy rozwiązania

---

[44] Zazwyczaj liczby pierwsze definiuje się jako liczby naturalne, które mają dokładnie dwa dzielniki (1 i samą siebie). Wtedy oczywiście liczba 1 nie jest liczbą pierwszą, więc w powyższym sformułowaniu w miejscu 1 powinna znaleźć się 2 (lub znak nierówności powinien być inny).

w postaci programu, jest na tyle dobrze zrozumiany, iż możliwe jest określenie wszystkich wymagań stawianych przed oprogramowaniem przed rozpoczęciem jego produkcji. Perspektywa ta pozwala na formalne ujęcie specyfikacji i weryfikacji, co z kolei umożliwia masową produkcję oprogramowania[45], podobnie do bardziej złożonych układów elektronicznych. Jednak perspektywa skoncentrowana na produkcie ma jedną zasadniczą wadę – nie pozwala na systematyczne badanie związków pomiędzy specyfikacją a rzeczywistym światem, które mają ogromny wpływ na zgodność produkowanego oprogramowania z oczekiwaniami zamawiających je ludzi.

Floyd wskazuje alternatywny sposób tworzenia specyfikacji, nazywany przez niego perspektywą zorientowaną na proces (ang. *process-oriented perspective*), w którym wymagania stawiane oprogramowaniu nie są nigdy statyczne. Zmieniają się one zarówno ze względu na zmianę wymagań zamawiającego, jak i na skutek opinii przyszłych użytkowników bądź też na wniosek zaangażowanych w proces programowania informatyków. Specyfikacja programu nie może być bowiem oddzielona od ludzi, dla których oprogramowanie jest produkowane. Komunikowanie się z osobą zamawiającą program oraz z przyszłymi jego użytkownikami jest konieczne, zatem specyfikacja powinna być wyrażana w języku naturalnym, a nie w zrozumiałym tylko dla fachowców formalizmie.

Co więcej, wbrew twierdzeniu powszechnemu wśród zwolenników paradygmatu matematycznego w programowaniu, specyfikacje formalne wcale nie pomagają w tworzeniu oprogramowania. Wręcz przeciwnie, ściśle formalne traktowanie programowania może szkodzić, zmniejsza bowiem efektywność pracy programistów. Zgodnie z paradygmatem matematycznym, programista powinien najpierw wyrazić rozwiązanie postawionego przed nim problemu w formalnym języku specyfikacji, później zaprogramować to samo rozwiązanie w języku programowania, a na koniec udowodnić, że oba te opisy są w jakimś sensie równoważne. Musi on wówczas stworzyć nie jeden, ale dwa formalne opisy tego samego problemu oraz udowodnić ich równoważność, podczas gdy kwestie związane z adekwatnością tych formalizmów do wyrażania rzeczywistych problemów nadal pozostają nierozwiązane.

Warto również zwrócić uwagę na wartość metody nieformalnej w tworzeniu oprogramowania. Pozwala ona bowiem na dyskutowanie, krytykowanie i definiowanie pojęć intuicyjnych. Oczywiście formalizacja może być w tym procesie pomocna – tak samo jak w dowodzeniu twierdzeń matematycznych – ale powinna być jedynie narzędziem pomocniczym intuicji, a nie ją zastępować. Naur (1982) pisze:

---

[45] Autor używa tu zwrotu „tłuczone masowo".

> [...] program powinien być wspomagany i specyfikowany przez dokumentację dowolnego rodzaju, sprawą nadrzędną w tworzeniu jego dokumentacji jest czytelność dla ludzi, którzy mają z nią do czynienia. Dla osiągnięcia tej czytelności (jasności) powinno się używać dowolnego systemu formalnego, nie jako celu samego w sobie, ale wtedy gdy wydaje się on pomocny autorom i czytelnikom (s. 452).

Jednak nawet jeśli wbrew poczynionym uwagom zgodzimy się co do konieczności i celowości tworzenia formalnych specyfikacji, to otwarty pozostaje problem pokazania zgodności tej specyfikacji z programem. Związany jest z nim często podawany argument przeciw weryfikacji programów, nazywany *argumentem ze zmienności*. Zgodnie z nim weryfikacja programów, rozumiana jako badanie ich zgodności ze specyfikacją, ma sens jedynie przy założeniu, że powstały one niezależnie oraz że zarówno program, jak i jego specyfikacja nie ulegają zmianom. Oba te założenia można łatwo obalić. Podczas procesu tworzenia oprogramowania wzajemne oddziaływania pomiędzy specyfikacją a programem są bardzo częste. Na przykład błędy znalezione w programie wymuszają zmiany w specyfikacji, a z kolei błędy w specyfikacji prowadzą do modyfikacji programu uwzględniających zmiany w samej specyfikacji.

DeMillo, Lipton i Perlis (1979) uważają, że nikt nie powinien traktować poważnie możliwości w pełni automatycznej weryfikacji skomplikowanych programów. Przypuśćmy jednak, że powstał program dokonujący automatycznego sprawdzania innych programów – i zastanówmy się, jak on działa. Oczywiście na początek należy założyć, że program ten jest poprawny, to znaczy, że funkcjonuje właściwie. Jest to jednak tylko założenie (ślepa wiara), gdyż nie można zweryfikować programu weryfikującego w nim samym. Programista wprowadza – zazwyczaj bardzo długą – specyfikację formalną oraz sam program i po dłuższym czasie otrzymuje – znowu bardzo długą – weryfikację i komunikat „ZWERYFIKOWANY". Oczywiście konieczne jest założenie, że komunikat ten nie jest wynikiem awarii systemu sprawdzającego lub jego części. Co z takiego komunikatu może odczytać programista? Wie on jedynie, że program jest formalnie, logicznie zgodny ze specyfikacją, nie wie natomiast, w jakim stopniu jest on wiarygodny, nie zna jego ograniczeń i nie zdaje sobie sprawy z tego, jak ograniczenia te mogą wpływać na jego działanie. Jeżeli z kolei pojawi się komunikat o tym, że program nie został zweryfikowany, to nie wiemy, dlaczego tak jest. Nie potrafimy zlokalizować błędu inaczej niż za pomocą metody prób i błędów, czyli poprawiając kod metodą „chybił trafił". Zdaniem autorów, można jednak dojść do wniosku – biorąc pod uwagę, że program jest przedmiotem materialnym wytworzonym przez człowieka oraz że każdy przedmiot o dużej złożoności

i dużym rozmiarze nie jest pozbawiony błędów – że komunikat nigdy nie będzie brzmiał „ZWERYFIKOWANY". Nie ma zatem sensu dążenie do zbudowania programów automatycznie weryfikujących programy. Dlaczego więc zwolennicy paradygmatu matematycznego w informatyce nalegają na pełną formalizację w procesie tworzenia oprogramowania?

Wydaje się, że głównym tego powodem jest przekonanie, iż pełna formalizacja jest niezbędna dla dowodów poprawności programów, która to poprawność jest absolutnym priorytetem dla informatyków.

Naur (1982) próbuje obalić tę tezę. Po pierwsze, pewne własności programów można udowodnić niezależnie od tego, czy zostały one sformułowane w sposób formalny, czy też nieformalny. Takie dowody podają, między innymi, Aho, Hopcroft i Ullman (1974). Są one bardzo podobne do tworzonych na co dzień dowodów matematycznych. Po drugie, nie jest prawdą, że poprawność programu jest priorytetem programowania, o czym można się przekonać każdego dnia pracy z komputerem. Większość używanych na co dzień programów zawiera błędy (wielu przykładów takich błędów dostarcza nam system Windows lub pakiet Microsoft Office), ale pomimo tego są one powszechnie stosowane i traktowane jako narzędzia użyteczne, a często nawet niezbędne.

W rozważaniach dotyczących własności programów ważną i często poruszaną kwestią jest rozróżnienie pomiędzy modelami matematycznymi a ich fizycznymi odpowiednikami. Kwestie tę analizują między innymi James Fetzer i Jon Barwise.

Fetzer (1991) twierdzi, że prawdziwe zdania matematyki mogą okazać się empirycznie fałszywe. Na poparcie swej tezy autor przytacza następujący przykład. Rozważmy matematyczne twierdzenie dotyczące liczb naturalnych:

$$2 + 2 = 4.$$

Nie może być ono fałszywe w obrębie teorii liczb naturalnych, ponieważ jego prawdziwość wynika z aksjomatów tej teorii. Jednak zastosowanie tego prawa do zjawisk fizycznych może okazać się fałszywe, na przykład:

2 krople wody+2 krople wody=4 krople wody.

Nie powinno się zatem zapominać o ograniczeniach, jakie rzeczywistość fizyczna narzuca podczas stosowania praw matematyki.

Można uznać, że nie istnieją prawdy matematyczne dotyczące obiektów fizycznych, ale nie ma wówczas sensu mówienie o możliwości weryfikacji programów, ponieważ nie jest możliwe stworzenie matematycznego dowodu dotyczącego własności danego programu i wykonanego na konkretnym obiekcie fizycznym (komputerze). Można też przyjąć, że sprawdzanie

własności programów, w tym ich weryfikacja metodami matematycznymi, jest możliwe, ale ma zawsze charakter relatywny, gdyż zamierzone zachowanie maszyny abstrakcyjnej można udowodnić dedukcyjnie, ale rzeczywiste zachowanie fizycznego urządzenia – komputera, na którym program jest wykonywany – będzie wiadome zawsze z pewną dozą niepewności.

Matematykę, która zdaniem Barwise'a opiera się na doświadczeniu fizycznego świata i służy lepszemu jego zrozumieniu[46], można wykorzystać do tego celu na dwa sposoby. Pierwszy to metoda aksjomatyczna polegająca na wyprowadzaniu kolejnych własności pewnej dziedziny z tych jej własności, które uznano za oczywiste (z aksjomatów). Drugi natomiast to modelowanie matematyczne, polegające na wykorzystaniu pewnych wcześniej stworzonych obiektów abstrakcyjnych do zbudowania modelu procesu lub obiektu fizycznego. Wnioski uzyskane w obu przypadkach są w pewnej mierze relatywne. W metodzie aksjomatycznej twierdzenia są bowiem prawdziwe, jeśli prawdziwe są aksjomaty. Z kolei w metodzie modelowania twierdzenia są faktami dotyczącymi modelowanego zjawiska tylko wtedy, jeśli sam model wystarczająco dokładnie oddaje modelowaną rzeczywistość. Nie ma przy tym możliwości udowodnienia prawdziwości poprzedników powyższych implikacji, ponieważ nie można udowodnić formalnie prawdziwości aksjomatów ani też adekwatności modelu badanego zjawiska. Nie dyskredytuje to jednak matematyki jako użytecznego narzędzia w dowodzeniu własności programów. Jeśli w ogóle powinno się mówić o programowaniu jako części matematyki, to powinna to być matematyka stosowana, a nie matematyka czysta. Matematyka stosowana nie gwarantuje, co prawda, wiedzy pewnej dotyczącej rzeczywistości fizycznej, może jednak prowadzić do uzasadnienia poprawnych przypuszczeń (Barwise, 1989).

Autorzy piszący na temat paradygmatu matematycznego w programowaniu bardzo często nie odróżniają modelu matematycznego od modelowanych obiektów. Błąd ten nazywany jest w literaturze „błędnym przekonaniem o identyfikacji" (ang. *the fallacy of identification*). Jeśli kończy się dowód jakiegoś twierdzenia, to powinno się zawsze pamiętać, że jest to dowód dotyczący modelu – a nie samej rzeczywistości. Barwise (1989) pisze: „Gdy umieszcza się końcowe 'QED' na końcu dowodu, to musimy uczynić krok wstecz i pamiętać, że identyfikacja jest tylko identyfikacją, a nie identycznością" (s. 848).

Formalne weryfikacje są zatem dowodami matematyki stosowanej, dowodami dotyczącymi modelu, a nie fizycznego urządzenia. Ich stosowalność do rzeczywistych systemów komputerowych nie jest ani bardziej, ani mniej

[46] Barwise zdaje sobie sprawę, że jest to stwierdzenie kontrowersyjne.

problematyczna niż stosowanie innych twierdzeń dotyczących matematycznych modeli pewnych zjawisk fizycznych. Dają one bowiem taki sam stopień pewności co do funkcjonowania rzeczywistości fizycznej.

W procesie weryfikacji programów to, co podlega sprawdzaniu, to tylko modele matematyczne rzeczywistych systemów komputerowych. Istnieje ryzyko związane z pominięciem w modelu formalnym pewnych własności maszyn fizycznych. Na przykład sposób reprezentowania liczb rzeczywistych może być znaczący dla realizacji działań na tych liczbach, a zależy on w zasadniczy sposób od charakterystyki konkretnych maszyn.

Co więcej, w przypadku weryfikacji całych systemów komputerowych, to znaczy programów i urządzeń, na których są one wykonywane, powinno się stworzyć matematyczne modele nie tylko dla komputerów, ale również dla otoczenia, w jakim one działają. Barwise (1989) pisze: „[…] pełny dowód poprawności systemu komputerowego musi składać się nie tylko z dowodu poprawności programu dotyczącego wiarygodnego modelu komputera, ale również z dowodu poprawności algorytmu dotyczącego środowiska, w którym znajduje się ten system" (s. 850).

Częścią tego środowiska są jego użytkownicy. Oczywiście stworzenie takiego modelu jest o wiele trudniejsze niż stworzenie modelu samego komputera – bardzo trudno jest bowiem przewidzieć zachowania człowieka. Wiele „luk" występujących w programach nie wynika z występujących w nich błędów, lecz z nieoczekiwanych, nieprzewidzianych przez programistów zachowań użytkowników. Co więcej, poprawny model powinien uwzględniać również inne czynniki mające wpływ na rzeczywiste zachowanie się programów, na przykład działanie systemu operacyjnego i sprzętu, na którym wykonuje się program (Hall, 1990).

Jeśli nawet przyjmiemy, że możliwe jest skonstruowanie odpowiednich modeli abstrakcyjnych, to trudno jest stwierdzić, czy modele te są dobre. Najczęściej przyjmuje się, że model jest dobry (adekwatny), gdy istnieje odwzorowanie z kluczowych cech modelu do odpowiadających im cech fizycznego świata, czyli – gdy istnieje izomorfizm pomiędzy modelem a tym, co jest modelowane. Naur (1992) twierdzi, że najważniejszymi zagadnieniami związanymi z poprawnym modelowaniem są: (1) uchwycenie relacji pomiędzy tymi aspektami świata, które nie mają ściśle zdefiniowanych istotnych własności, a formalnie zdefiniowanymi elementami modelu (ang. *modeling insight*), oraz (2) znalezienie sposobu na połączenie dostępnych elementów modelu w taki sposób, by otrzymać model o odpowiednich własnościach (ang. *model building insight*). Tylko wtedy można otrzymać adekwatny model fragmentu rzeczywistości. Bardzo rzadko można jednakże podać dowód takiej adekwatności. Najczęściej istnieje tylko empiryczne, eksperymentalne potwierdzenie, że model jest dobry – na przykład gdy

pozwala on na skuteczne przewidywanie faktów dotyczących modelowanej dziedziny. Tak więc w metodach badania programów nie można ograniczyć się do metod formalnych, z pominięciem eksperymentu.

Z przedstawionych powyżej rozważań można wysnuć błędny wniosek, że krytyka paradygmatu matematycznego w programowaniu ogranicza się do argumentów przeciw możliwości formalnej weryfikacji programów. Tymczasem wielu autorów podważa również samo rozumienie programu komputerowego jako obiektu matematycznego.

Fetzer (1988, 1991) twierdzi, że pomimo iż rozumienie programów jako funkcji ze zbioru stanów początkowych (lub „wejść") w zbiór stanów końcowych (lub „wyjść") – analogiczne do wynikania w dowodach matematycznych, jako funkcji ze zbioru przesłanek do zbioru wniosków[47] – ma wiele zalet, to programy komputerowe różnią się od dowodów. Co prawda programy, podobnie jak dowody matematyczne, są obiektami syntaktycznymi i jako takie jawią się jako całkowicie formalizowalne byty, dla badania których właściwe są procedury formalne. Jednak – pod niektórymi względami – bardziej przypominają teorie naukowe niż dowody matematyczne. Programy, w przeciwieństwie do dowodów matematycznych, mają znaczenie semantyczne, podobnie jak teorie naukowe, ponieważ polecenia tworzące program zastępują operacje i procedury, które mogą być wykonane przez rzeczywistą maszynę. Fetzer (1991) pisze: „Linie składające się na program, podobnie jak zdania tworzące teorię, ostatecznie dla użytkowników tych programów i tych teorii wydają się reprezentować inne rzeczy" (s. 202).

Jednak teorie naukowe różnią się od programów komputerowych, ponieważ w przeciwieństwie do nich nie posiadają zdolności przyczynowych. Programy mogą wpływać na zachowanie się maszyn, na których są wykonywane, natomiast teorie naukowe nie wpływają bezpośrednio na rzeczywistość, lecz tylko ją opisują.

Cechy dowodów matematycznych, teorii naukowych oraz programów komputerowych Fetzer zestawił w postaci tabeli:

Tabela 6. Cechy dowodów matematycznych, teorii naukowych i programów komputerowych

	Dowody matematyczne	Teorie naukowe	Programy komputerowe
Obiekty syntaktyczne	Tak	Tak	Tak
Znaczenie semantyczne	Nie	Tak	Tak
Zdolności przyczynowe	Nie	Nie	Tak

Źródło: Fetzer, 1991, s. 202

[47] Por. paragraf 2.1.2.

Z powyższego zestawienia wynika, że jeśli programy komputerowe posiadają zarówno zdolności przyczynowe, jak i znaczenie semantyczne, to metody oceny ich poprawności powinny wykraczać zarówno poza metody formalne, jak i poza metody używane w teoriach naukowych, takie jak obserwacja i eksperyment. Fetzer uważa, że należy stosować w tym celu prototypowanie i testowanie, ponieważ są one związane bezpośrednio z wykonaniem programów przez maszyny. Twierdzi przy tym, że wprowadzone przez zwolenników metod formalnych w informatyce rozumienie programu komputerowego jako abstrakcyjnych operacji na symbolach całkowicie pomija istotne rozróżnienie pomiędzy programem jako bytem abstrakcyjnym a programem jako wykonywalnym kodem[48].

Wielu informatyków opowiada się, podobnie jak Fetzer i Naur (1966), za przyjęciem w procesie tworzenia oprogramowania wszystkich skutecznych metod i nieograniczaniem się do metod formalnych. Blum (1989) twierdzi, że w przypadku dużych programów bardzo użyteczne jest prototypowanie[49], czyli budowanie prototypu programu – który może być dość daleki od ostatecznego produktu – w celu jego przetestowania. Jest ono konieczne, jeśli rozumie się proces tworzenia oprogramowania jako złożenie: (1) abstrakcji, dla stworzenia dokładnej specyfikacji problemu z jakiejś rzeczywistej dziedziny, oraz (2) urzeczywistnienia (reifikacji), przekształcenia specyfikacji w program.

Centralną rolę w tym procesie odgrywa specyfikacja, która jest zazwyczaj niekompletna. Trudno jest bowiem zamawiającemu program podać już na początku wszystkie swoje wymagania na tyle precyzyjnie, by nie trzeba było specyfikacji modyfikować. Trudność ta może wynikać między innymi z przejścia od pojęciowego myślenia o problemie do myślenia formalnego. Według Bluma (1989) „proces tworzenia oprogramowania nie jest tylko logiczną derywacją produktu z ustalonej specyfikacji" (s. 13).

A zatem w procesie urzeczywistniania konieczne jest ciągłe ulepszanie specyfikacji, gdyż dzięki niemu uzyskuje się głębsze zrozumienie problemu, a w efekcie lepszy program. W procesie abstrakcji pomija się bowiem pewne szczegóły, które mogą okazać się istotne i których dodanie stanie się później niezbędne. Blum sugeruje, że szybkie prototypowanie[50] (ang. *rapid*

---

[48] Por. paragraf 2.1.1.

[49] Prototypowanie jest czasami przedstawiane jako paradygmat alternatywny do paradygmatu matematycznego w programowaniu. Blum twierdzi jednak, że jednoczesne wykorzystanie prototypowania i metod formalnych może być niezwykle owocne w procesie tworzenia oprogramowania oraz że mogą one współistnieć w obrębie tego samego paradygmatu.

[50] Więcej szczegółów dotyczących tej techniki znaleźć można w pracy Bluma *Formalism and Prototyping in the Software Process* (Blum, 1989).

prototyping) jako „technika przedefiniowywania początkowej specyfikacji" może znacznie ulepszyć ten proces.

W wielu pracach dotyczących analizy paradygmatu matematycznego w informatyce zwraca się uwagę na praktykę programowania. Na przykład Scherlis (1989) zauważa konieczność oddzielenia jakości programów (jako rezultatów procesu programowania) od metod wykorzystywanych podczas ich tworzenia. Programista praktyk powinien móc używać wszelkich metod, które prowadzą go do celu, jakim jest stworzenie dobrego programu. Natomiast dowód poprawności programu, rozumiany jako jego związek ze specyfikacją, należy rozumieć jako rodzaj jego dokumentacji. Co więcej, w procesie tworzenia specyfikacji i dokumentacji programów powinny być wykorzystywane zarówno komponenty formalne, jak i nieformalne. Ograniczenie się tylko do metod formalnych natomiast można zastosować w celu sprawdzenia kluczowych własności programu. Jednakże nawet wtedy powinno się dopuścić używanie metod nieformalnych (chociaż mniej wiarygodnych) – a to w celu zwiększenia przekonania co do innych (niekluczowych) cech systemu. Scherlis (1989) pisze: „Efektem będzie dowodzenie małych twierdzeń o dużych systemach, a nie dużych twierdzeń o małych systemach" (s. 1047).

Warto zauważyć, że idea dowodzenia poprawności programu przed jego wykonaniem, jaką postulują często zwolennicy paradygmatu matematycznego, w większości praktycznych problemów nie ma zastosowania (Hamming, 1969). Tworzenie takich dowodów ma sens jedynie wtedy, gdy chce się stworzyć kolejny kompilator nowego, zdefiniowanego matematycznie języka programowania. Jednak w wielu przypadkach, jeśli nie w ich większości, w programowaniu – podobnie zresztą jak w matematyce – istnieje przepaść pomiędzy tym, co powinno się zrobić (tym, co się postuluje), a tym, co można wykonać w praktyce.

Większość skomplikowanych programów działa zazwyczaj poprawnie – choć przyznać trzeba, że niekoniecznie w najbardziej elegancki sposób – choć bez wykorzystania metod formalnych w procesie ich tworzenia. Metody inżynieryjne pozwalają bowiem na tworzenie dość wiarygodnych systemów w rozsądnym czasie i po rozsądnych kosztach. Kluczowe jest tu odróżnienie słowa „rozsądnie" od „optymalnie" czy „idealnie" (Winograd, 1989).

Rzeczywiste programy bardzo często mogą nie mieć charakterystyki matematycznej, wynikającej z używanych aksjomatów. Ponadto błędy w dowodach formalnych w matematyce są tak samo częste jak błędy w metodach inżynierii. Potrzebne jest zatem, poza stosowaniem metod formalnych w programowaniu, również testowanie, które może rozwiązać oba te problemy. Właściwie zaprojektowane testowanie może dostarczyć

informacji na temat prawdopodobieństwa wystąpienia błędów w kodzie oraz na temat prawdopodobieństwa ich ujawnienia w czasie wykonania programów. Pozwala ono również na lepsze dopasowanie modelu matematycznego do problemu rozwiązywanego przez program, czego nie zapewni formalizacja. Parnas (1989) pisze: „Nie ma dziedziny inżynierii, w której testowanie i uzasadnienie matematyczne traktowane są jako alternatywy. Powszechnie twierdzi się, że dopełniają się one i że obie są potrzebne" (s. 1405).

Po przeanalizowaniu uwag informatyków na temat tworzenia rzeczywistych programów komputerowych można stwierdzić, że programowanie nie jest współcześnie działalnością matematyczną, choć być może kiedyś taką będzie. Obecnie jedynie „[l]udzie, którzy nie znają ani programowania, ani matematyki (tj. prawie wszyscy), uznają za oczywiste, że programowanie jest jak matematyka" (Van Emden, 1989, s. 1407).

2.3. Podsumowanie

Rozdział ten poświęcono jednej z podstawowych działalności informatyków, czyli programowaniu oraz jego wytworom – programom komputerowym. Programowanie jest to proces złożony i wieloetapowy. Jego początek stanowi stworzenie specyfikacji, opisu problemu, jaki program ma rozwiązywać; potem następuje wybór (lub stworzenie) algorytmu, czyli metody jego rozwiązania. Kolejnym etapem jest implementacja tego algorytmu w wybranym języku programowania, najczęściej w jednym z języków wysokiego poziomu. Procesor komputera wykonuje wyłącznie operacje zapisane w kodzie maszynowym, tak więc konieczne jest tłumaczenie – kompilacja lub interpretacja – kodu źródłowego programu do postaci binarnej. Tak powstały program poddawany jest sprawdzeniu, bądź to eksperymentalnemu (testowaniu), bądź też – formalnemu, polegającemu na stworzeniu matematycznego dowodu jego poprawności. Zawarte w tym rozdziale rozważania unaoczniają nam, jak ciekawym obiektem badań dla filozofa jest ów proces programowania oraz powstały w jego wyniku byt, nazywany programem komputerowym.

Paragraf 2.1.1 poświęcony został jednemu z najczęściej dyskutowanych problemów ontologicznych związanych z programami komputerowymi, a mianowicie ich dualnej naturze. Jest ona wyrażana stwierdzeniem, nazywanym dwuznacznością Fetzera, że program można rozumieć zarówno jako ciąg instrukcji zapisanych w języku programowania (byt abstrakcyjny), jak i jako proces, czyli wykonanie kodu źródłowego na maszynie fizycznej. Rozróżnienie to dobrze oddaje taksonomia obiektów informatyki stworzona przez A.H. Edena oraz R. Turnera (2006).

Wyróżniają oni trzy kategorie bytów, z którymi mają do czynienia informatycy: *Hardware* (maszyny nazywane komputerami), *Metaprogramy* (formuły zapisane w języku rachunku predykatów, opisujące programy) oraz *Programy*. Podział obiektów ostatniej kategorii na dwie grupy (*Skrypty* i *Boty*) odpowiada dwuznaczności Fetzera. Obiekty kategorii *Boty* to grupa bytów czasowych (procesów), które powstają w wyniku uruchomienia skryptów w konkretnych warunkach fizycznych. Z kolei *Skrypty* definiowane są jako zawierające poprawnie sformułowane instrukcje dla pewnej klasy komputerów cyfrowych, to znaczy jako byty abstrakcyjne. Wśród skryptów autorzy wyróżniają dwie podkategorie: *Kody Maszynowe*, zapisane w języku maszynowym, i *Kody Zródlowe* – skrypty zapisane w języku programowania wysokiego poziomu.

Przyjmując, podobnie jak Eden i Turner, dwuznaczność Fetzera jako cechę charakterystyczną programów, należy odpowiedzieć na pytanie, jaki jest związek programów-napisów z ich wykonaniami. I tak można by przyjąć, że program rozumiany jako proces oraz jego zapis są różnymi manifestacjami tego samego obiektu. Bardziej intuicyjne jednakże wydaje się stwierdzenie, że programy-napisy powodują (wywołują) procesy zachodzące w komputerze. W paragrafie 2.1.1 przedstawiono próby odpowiedzi na związane z tym poglądem pytanie o naturę takiego związku przyczynowego: propozycję traktowania programu komputerowego jako konkretnej abstrakcji (pochodzącą od Colburna) oraz – jako obiektu quasi-partykularnego (na wzór utworu muzycznego). Wydaje się jednak, że dużo więcej zwolenników ma rozumienie programu jako bytu matematycznego (któremu poświęcono paragraf 2.1.2).

Traktowanie informatyki jako nauki formalnej, nowego działu matematyki, jest jednym z najszerzej dyskutowanych i krytykowanych w literaturze poglądów filozoficznych z nią związanych. Zwolennicy tego poglądu twierdzą, że zarówno programy rozumiane jako napisy (opisujące zachowanie komputerów, na których są wykonywane), jak i programy-procesy to obiekty matematyczne. Te drugie są bowiem opisywane przez wyrażenia matematyczne, jakimi są programy-napisy.

W paragrafie 2.1.2 dokonano analizy statusu wyrażeń stanowiących programy na przykładzie instrukcji podstawienia występującej we wszystkich językach programowania. Uznanie ich za zdania analityczne (właściwe dla nauk formalnych, w tym dla matematyki) byłoby ważnym argumentem na rzecz traktowania programów jako obiektów matematycznych. Wyrażenia opisujące operację podstawienia można interpretować na różne sposoby: jako rozkazy, które nie będąc zdaniami w sensie logicznym, nie są ani analityczne, ani syntetyczne, albo jako zdania oznajmujące, rozumiane dwojako. Jeśli dotyczą one fizycznych miejsc w pamięci, to operacja pod-

stawienia jest tylko przypuszczeniem co do wyniku wykonania programu i jako taka jest zdaniem syntetycznym, którego prawdziwość zależy od rzeczywistości, na przykład – od poprawności kompilacji tej instrukcji czy też od właściwego działania komputera. Jeśli z kolei rozumiemy instrukcję podstawienia jako odniesienie do stanu maszyny abstrakcyjnej, to jest ona pewnego rodzaju założeniem (podobnym do założenia twierdzenia matematycznego rozpoczynającego się od „niech…", ale bez późniejszego „…wtedy"), które nie jest ani analityczne, ani syntetyczne. A zatem operacja podstawienia, podobnie jak inne wyrażenia języków programowania, jest albo zdaniem syntetycznym, albo też jest podobna do fragmentów wyrażeń matematycznych (których nie można traktować jako zdań analitycznych). Tym samym nie można stwierdzić, że programy są obiektami matematycznymi w klasycznym sensie.

Jeśli jednak, wbrew wielu wątpliwościom, przyjmiemy, że programy są obiektami matematycznymi, to powstaje pytanie, jakiego rodzaju są to obiekty. Można je potraktować (por. Fetzer, 1991) jako funkcje działające ze zbioru „wejść" (danych) do zbioru „wyjść" (możliwych wyników), bądź też rozumieć je „metamatematycznie", jako odpowiedniki twierdzeń matematycznych (Scherlis, Scott, 1983).

Suber (1998) zaproponował – omówioną w paragrafie 2.1.3 – definicję programu komputerowego jako cyfrowego wzorca, rozumianego jako tablica komórek (miejsc) zawierających jeden z dwóch możliwych symboli. Podał on również szereg zasad, założeń ontologicznych z nią związanych. Twierdzi między innymi, że każdy wzorzec analogowy może zostać odtworzony we wzorcu cyfrowym (zasada cyfrowa) oraz że wzorce cyfrowe i analogowe wyczerpują dziedzinę wzorców (zasada wyczerpywania). Każdy rodzaj wzorca może więc być programem, bez względu na sposób rozumienia samego wzorca: jako bytu materialnego czy też może w sensie platońskim – jako idei.

Aby umożliwić odróżnienie programów od losowego układu zer i jedynek (tj. szumu), autor proponuje przyjęcie zasady bezszumowości, głoszącej, że żaden wzorzec nie jest szumem dla wszystkich możliwych języków programowania i wszystkich możliwych maszyn. Wtedy jednak przy odpowiedniej interpretacji każdy układ zer i jedynek można rozumieć jako program. Dla uniknięcia tego problemu Suber wprowadza dodatkowe warunki na wzorce, które mają stanowić programy: ich czytelność dla maszyny oraz – wykonywalność. Modyfikacja ta nie rozwiązuje jednak problemu odróżnienia programu od szumu, ponieważ wzorzec będący szumem może zostać wykonany przez maszynę, tyle tylko, że zostanie wykonany źle. Wydaje się, że dla odróżnienia różnych wykonań programów konieczne jest odwołanie się do celu programisty. Poprawnie działające programy realizują bowiem

cele swoich autorów, natomiast kody losowe lub błędne – nie. Pojęcie celu programisty jednakże nie jest łatwe do zdefiniowania, szczególnie jeśli zgodzimy się z istnieniem „programów naturalnych" – wpisanych w przyrodę, dla których trudno jest określić ich twórcę. Ponadto wymóg czytelności wzorców powinien zostać sprecyzowany, na przykład jako stwierdzenie, że wzorzec *można* wyrazić w odpowiedniej formie fizycznej, by móc uznać za program również jego zapis na kartce papieru, a ten przecież nie jest czytelny dla maszyny. Jeśli jednak przyjmiemy zaproponowaną przez Subera zasadę dostrzegalności, która mówi, że wszystkie wzorce spełniają wymaganie fizyczne, a także jeśli uznamy zasadę cyfrową, wówczas każdy wzorzec spełnia założenie o czytelności i wykonywalności. Widzimy więc, że dodanie zasad jest zbędne, nie rozwiązuje bowiem problemu. Programem nadal może być każdy wzorzec – i tak, na przykład może nim być układ gwiazd w Drodze Mlecznej.

Z definicją podaną przez Subera związane są jeszcze dwa inne ważne problemy ontologiczne: rozróżnienie wzorca jako danych i wzorca jako programu oraz określenie kryteriów równości programów. Rozwiązanie pierwszego z nich autor widzi w stwierdzeniu, że wzorce-programy to te, które uruchamiane są wcześniej i które pełnią funkcję kontrolną względem wzorców-danych. Bardziej problematyczne jest odróżnienie programów na podstawie wzorców. Jeśli bowiem rozumieć wzorzec w terminach czysto syntaktycznych, to różne wzorce (różniące się choćby jednym bitem) są różnymi programami. Jednak kod programu w języku maszynowym, powstający przez kompilację kodu napisanego w języku programowania wysokiego poziomu, to różne wzorce będące tym samym programem. Suber uważa, że do identyfikacji programów potrzebne jest przejście od składni do semantyki (uwzględniającej pojęcie celu programisty); dodaje przy tym do listy zasad kolejną (którą nazywa zasadą pitagorejską), głoszącą, że składnia „budzi" semantykę, co pozwala na stwierdzenie, że istotą programu jest sam wzorzec.

Zaproponowana przez Subera definicja programu jako wzorca budzi jednak wątpliwości. Autor dodaje do niej liczne zasady, zazwyczaj bez ich uzasadnienia, które są silnymi założeniami ontologicznymi. Co więcej, używane pojęcie „wzorca cyfrowego", bez bliższego określenia jego natury, wydaje się nie mniej problematyczne niż samo pojęcie programu. Jeśli rozumiemy wzorce jako obiekty matematyczne, to definicja Subera jest tylko kolejną, nie do końca przemyślaną i niedostatecznie uzasadnioną wersją twierdzenia, że programy są obiektami matematycznymi.

Jednym z najważniejszych problemów ontologicznych związanych z programami komputerowymi jest – rozważa je również Suber – określenie kryteriów ich identyczności. Jak już wspomniano, utożsamienie programu

z jego fizyczną reprezentacją zmusza nas do rozróżnienia programu zapisanego w języku programowania wysokiego poziomu i efektu jego kompilacji (lub interpretacji). Trzeba więc szukać innego kryterium identyczności programów, takiego, które nie koncentruje się wyłącznie na sposobie ich zapisu. Przydatna okazuje się semantyka języków programowania.

Języki programowania – traktowane jako zbiór definicji i reguł – mają swoją gramatykę (określającą poprawność syntaktyczną wyrażeń) oraz semantykę (definiującą znaczenie występujących w nich konstrukcji). Semantyki języków programowania pomagają w tworzeniu nowych języków, służą do ich klasyfikowania, badania poprawności oraz do określania kryteriów ich identyczności. Współcześnie wyróżnia się trzy podstawowe typy semantyk formalnych: operacyjne, denotacyjne oraz aksjomatyczne. W paragrafie 2.1.4 poddano analizie przykładowe semantyki operacyjne i denotacyjne, bazujące na obiektach matematycznych (takich jak zbiory czy funkcje), ze zwróceniem szczególnej uwagi na wprowadzane przez nie kryteria identyfikacji programów.

Semantyki operacyjne opisują każdą strukturę języka przez czynności, jakie jej odpowiadają na maszynie abstrakcyjnej, tłumacząc wyrażenia tego języka na ciąg instrukcji dla niej. Semantyki denotacyjne natomiast charakteryzują znaczenie zapisów w danym języku za pomocą funkcji matematycznych (wykorzystując najczęściej logikę, teorię mnogości oraz teorię kategorii), określających wartość danego wyrażenia (jego sens). Turner (2007) twierdzi, że semantyki operacyjne i denotacyjne nie różnią się w istotny sposób, ponieważ obie wprowadzają interpretacje matematyczne. Można jednak wskazać takie programy (por. paragraf 2.1.4), które pewna semantyka denotacyjna może utożsamiać (gdyż obliczają one tę samą funkcję), semantyka operacyjna natomiast może je rozróżniać, jeżeli brać pod uwagę poszczególne kroki obliczeń. Która z nich jest zatem odpowiednia do określenia równości programów?

Niektórzy badacze twierdzą (Turner, Eden 2011), że każda semantyka języka programowania wyznacza kryterium równości napisanych w tym języku programów, ponieważ dwa programy są równe wówczas, jeśli mają tę samą wartość semantyczną. Wydaje się jednak, że semantyki służące do utożsamiania programów powinny mieć pewne szczególne cechy – zgodność i pełność – określane łącznie jako pełna abstrakcyjność. Takie semantyki spełniają bowiem prawo Leibniza i oddają strukturę programów, abstrahując od ich szczegółów notacyjnych czy implementacyjnych. Odzwierciedlenie tego prawa zawarte jest w pojęciu obserwacyjnej równoważności programów, którą, z filozoficznego punktu widzenia, można rozumieć jako rodzaj funkcjonalizmu. Turner (Turner, Eden 2011) definiuje ją następująco: Dwa programy P i Q są *obserwacyjnie równoważne* wtedy

i tylko wtedy, gdy we wszystkich kontekstach C[•], w których C[P] jest poprawnym programem, również C[Q] jest poprawnym programem o tej samej wartości semantycznej. Oczywiście w praktyce nie jest możliwe sprawdzenie zachowania programu we wszystkich możliwych kontekstach i dlatego do pojęcia obserwacyjnej równoważności należy podchodzić z pewną ostrożnością.

Semantyka może również pełnić funkcję normatywną, pomagając w projektowaniu języków programowania posiadających odpowiednie własności metateoretyczne, na przykład przejrzystość referencyjną (wzajemną zastępowalność termów stojących za tymi samymi obiektami). Przyczynia się ona do większej kontroli poprawności programu, ułatwia bowiem jej dowodzenie.

Czynnikiem ułatwiającym tworzenie programów, poza przejrzystością referencyjną samego języka programowania, jest odpowiednia forma specyfikacji rozumianej jako każde przedstawienie wymagań stawianych przed programem. Tworzone w praktyce specyfikacje różnią się poziomem szczegółowości: od opisów wyrażonych w języku naturalnym (potocznym) aż po specyfikacje czysto formalne, rozumiane jako modele konceptualne. Nie ma powszechnej zgody nie tylko co do tego, jaka jej forma jest najlepsza, ale również w kwestii tego, jakiego rodzaju specyfikacje są wykorzystywane w praktyce.

Wielu informatyków akceptuje wymóg tworzenia specyfikacji formalnych, a niektórzy z nich postulują nawet o to, by były one wykonywalne. Te ostatnie są zazwyczaj rozumiane jako modele funkcjonalne programów (lub urządzeń) i jako takie ułatwiają tworzenie poprawnych programów. W szczególności pomocne są we wczesnym wykrywaniu i poprawianiu błędów w ich projektach. Ponadto umożliwiają natychmiastowe sprawdzenie tych fragmentów oprogramowania, których nie można sprawdzić w sposób formalny – na przykład interfejsu użytkownika. Specyfikacje wykonywalne mogą również służyć jako prototypy, umożliwiające eksperymentowanie z różnymi wymogami stawianymi przed programem, stając się znakomitym narzędziem komunikacji pomiędzy zamawiającym program klientem a tworzącym go zespołem programistycznym.

Wykorzystanie w procesie tworzenia programów specyfikacji formalnych, a w tym wykonywalnych, rodzi pytanie o różnice pomiędzy nimi a opisywanymi przez nie programami. Najczęściej podawanym kryterium odróżniania jest poziom szczegółowości: program zawiera bardzo szczegółowy opis działania komputera (instrukcja po instrukcji), natomiast specyfikacja opisuje tylko związek pomiędzy wejściem a wyjściem. Takie ujęcie nie ma jednak zastosowania w przypadku wielu specyfikacji formalnych, zapisanych za pomocą specjalnych języków, których poziom szczegółowości

zbliżony jest do złożoności programów. Inne kryterium, wykorzystujące pojęcie funkcji obiektu, wyraża się stwierdzeniem, że specyfikacje mają charakter deskryptywny (opisują obiekt), w odróżnieniu od programów, które są imperatywne. Jednak rozróżnienie to jest właściwe tylko w przypadku programowania imperatywnego, a nie jest to jedyny obowiązujący paradygmat. Nie wszystkie programy są imperatywne, co więcej, nie wszystkie specyfikacje są deskryptywne (gdyż niektóre zawierają dodatkowe wymagania związane na przykład z rodzajem instrukcji, jakich należy użyć w programie). Tak więc rozróżnienie deskryptywne/imperatywne nie będzie odpowiednim kryterium. Kolejną opisaną w literaturze ideą jest twierdzenie, że program komputerowy – w przeciwieństwie do specyfikacji – może zostać skompilowany i wykonany. Z takim ujęciem wiąże się jednak wiele niejasności i trudności, które zostały opisane w paragrafie 2.1.5. Jeśli zakładamy, że z możliwością „kompilacji" wiąże się fizyczne istnienie kompilatora, to status obiektów może zmieniać się w czasie: coś, co dziś jest specyfikacją (ponieważ nie istnieje dla niego kompilator), niebawem może stać się programem. Co więcej, istnienie wspomnianych wcześniej specyfikacji wykonywalnych podważa zasadność takiego twierdzenia.

Wydaje się, że najbardziej odpowiednie kryterium odróżnienia specyfikacji od programów związane jest z intencjami ich tworzenia (Fuchs, 1992). Specyfikacja, w przeciwieństwie do programu, jest bowiem nie tylko opisem działania artefaktu, wprowadza bowiem również kryteria poprawności tego działania, a to znaczy, że pełni funkcję normatywną. Czym jest tak rozumiana specyfikacja?

Można ją traktować jako definicję warunkową artefaktu (Turner, 2011), z zastrzeżeniem, że nie każda taka definicja jest specyfikacją, ponieważ musi ona posiadać siłę normatywną w procesie konstrukcji artefaktu. Można również twierdzić, że specyfikacja jest odpowiednikiem teorii naukowej, którą informatycy poddają sprawdzeniu przez skonstruowanie artefaktu przez nią opisywanego. Specyfikacje nie są jednak opisami świata, nie podają wiedzy na temat tego, jaki on jest. I dalej, nie są one wyjaśnieniami naukowymi, a także nie przewidują, że artefakt będzie działał poprawnie. Opisują tylko znaczenie „poprawnego działania". Nie można ich zatem nazywać teoriami naukowymi w klasycznym sensie. Oprócz ujęcia specyfikacji jako definicji czy teorii naukowych spotyka się też twierdzenie, że są one opisami funkcjonalnymi obiektów – to jest mogą mieć tę samą formę, zbliżony poziom szczegółowości, jak również mogą być opisane podobnym językiem (formalnym lub nie). Powstaje zatem pytanie, czy nie są to pojęcia tożsame, a jeśli nie, to jak odróżnić specyfikację od opisu funkcjonalnego. Jedyną istotną różnicę stanowi czas ich powstania (Turner, 2011). Jeśli bowiem opis powstał przed konstrukcją obiektu, to mamy do czynienia ze

specyfikacją (posiada ona charakter normatywny), w przeciwnym wypadku – będzie to tylko opis jego funkcjonowania.

Równie ciekawym tematem badań dla filozofa, związanym z procesem tworzenia programów, jest implementacja. Już samo to pojęcie jest wieloznaczne. W informatyce pojawia się ono przynajmniej w trzech różnych kontekstach: zapisu algorytmu w języku programowania (program jest implementacją algorytmu), w sytuacji, gdy programy są implementowane w języku maszynowym, oraz – w związku ze strukturami danych (gdy są one implementacją abstrakcyjnych typów danych). Na szczególną uwagę zasługuje definicja implementacji (Rapaport, 1999) jako interpretacji semantycznej, ponieważ takie ujęcie ułatwia wyjaśnienie natury symulacji komputerowych. Jednak analizy różnych informatycznych przykładów implementacji przedstawione w paragrafie 2.1.6 wykazują, że niektóre z nich nie spełniają stawianego przed interpretacjami semantycznymi wymogu – nie podają kryterium poprawności. A zatem nie wszystkie implementacje są interpretacjami semantycznymi, gdyż – wbrew opinii Rapaporta – nie każdy związek pomiędzy dwiema dziedzinami, w której jedna jest używana, by rozumieć lub zinterpretować drugą, jest interpretacją semantyczną.

Przedstawione w podrozdziale 2.1 rozważania pokazują, jak wiele pojęć związanych z programem komputerowym oraz z procesem jego tworzenia wymaga pogłębionej refleksji filozoficznej. Trudności dotyczą jednak nie tylko kwestii ontologicznych, wiążą się one także ściśle z zagadnieniami epistemologicznymi.

Jednymi z najszerzej dyskutowanych kwestii epistemologicznych dotyczących pojęcia programu komputerowego są metody badania jego poprawności. Już samo rozumienie pojęcia „poprawny" w odniesieniu do programu komputerowego budzi wiele kontrowersji. Najczęściej przez zwrot „poprawny program" rozumie się albo jego zgodność ze specyfikacją, albo też – program rozwiązujący stawiany przed nim problem. W paragrafie 2.2.1 zbadano kwestie związane z dopuszczalnymi metodami sprawdzania poprawności programów.

Zwolennicy tak zwanego paradygmatu matematycznego w informatyce twierdzą, że programy są bytami matematycznymi i dlatego dla pokazania poprawnego ich działania należy tworzyć formalne dowody ich zgodności ze specyfikacją. Dopuszcza się przy tym zastosowanie programów weryfikujących, tworzących tak zwane weryfikacje. Pogląd ten jest szeroko krytykowany w literaturze przedmiotu. Niektórzy badacze twierdzą bowiem, że nie można zweryfikować matematycznie poprawności programu, ponieważ jest on elementem systemu przyczynowego, którego nie da się sprawdzić metodami formalnymi. Możliwe jest tylko udowodnienie poprawności algorytmów wykorzystanych przy jego tworzeniu. To, co można stwierdzić, to

wiarygodność systemu komputerowego, a do tego celu wystarczy obserwacja jego wykonania, czyli testowanie.

Testowanie programów, omówione w paragrafie 2.2.2, polega na uruchomieniu programu dla reprezentatywnego zbioru danych i obserwacji jego wyników. Oczywiście w przypadku większości nawet prostych programów nie jest realne sprawdzenie ich działania dla wszystkich możliwych danych początkowych. Tak więc, jak zauważył Dijkstra (1989), testowanie programu może wykazać obecność błędów – ale nie ich brak. Nie podważa to jednak celowości stosowania tej metody badania, ponieważ może ona zwiększyć przekonanie co do poprawnego działania danego programu (co jest całkowicie zgodne z metodologią nauk eksperymentalnych).

W paragrafie 2.2.2 omówiono również konteksty, w których używa się w informatyce terminu „testowanie". Najczęściej testuje się program w celu sprawdzenia jego poprawności rozumianej jako zgodność z formalną specyfikacją. Można również uruchomić program, by stwierdzić, czy system sprzętowo-programowy, na którym jest on wykonywany, funkcjonuje właściwie. Oczywiście, oba te rodzaje testowania są ściśle ze sobą związane. Z kolei gdy tworzy się program komputerowy w celu rozwiązania jakiegoś problemu, to testuje się go, w celu sprawdzenia, czy spełnia on stawiane przed nim wymagania – czy rozwiązuje problem. Kluczową rolę odgrywa wówczas metoda rozwiązania, czyli algorytm[51]. Colburn (2000) wyjaśnia rolę algorytmu w komputerowym rozwiązywaniu problemów poprzez analogię z metodologią nauk przyrodniczych. Twierdzi on, że pisanie programów jest w informatyce tym samym, co konstruowanie modeli w naukach przyrodniczych – i że informatyka jest nauką eksperymentalną. Podobny sposób ujmowania informatyki ma jednak wielu przeciwników, którzy wskazują między innymi na trudności z określeniem różnic pomiędzy programem a algorytmem.

Programy tworzy się również i uruchamia w celu sprawdzenia pewnej hipotezy naukowej w komputerowym modelu rzeczywistości. Stanowią one wówczas tylko narzędzia w procesie sprawdzania (podobnie jak aparatura badawcza) i służą do testowania hipotez oraz do odkrywania nieznanych dotąd praw natury. Powyższe ujęcie testowania związane jest z tak zwanym paradygmatem naukowym, czyli stwierdzeniem, że programowanie (lub nawet cała informatyka) jest nauką przyrodniczą.

Bliski związek testowania programów z naukami eksperymentalnymi potwierdza analiza używanych w nim rozumowań, przedstawiona przez Colburna (2000). Twierdzi on, że argumentacje zawarte w testowaniu to rodzaj rozumowań indukcyjnych, które nazywa się podważalnymi (ang.

[51] Sprawdzanie poprawności algorytmu jest blisko związane z kolejnym celem testowania programu – z badaniem, czy zaproponowane w specyfikacji rozwiązanie jest poprawne.

defeasible), ponieważ odnoszą się one do założenia o „typowości" nieobserwowalnych przypadków (niesprawdzonych podczas testowania wejść programu). Owa podważalna natura wnioskowań zawartych w testowaniu programów nie gwarantuje ich właściwego działania w sposób absolutny. Dlatego też niektórzy naukowcy twierdzą, że testowanie powinno zostać zastąpione lub przynajmniej uzupełnione dowodami formalnymi.

Postulat przeprowadzania formalnych dowodów poprawności programów, rozumianej jako jego zgodność ze specyfikacją, związany jest ze wspomnianym wcześniej paradygmatem matematycznym w informatyce. Prace nad formalną weryfikacją trwały od lat 60. XX wieku, a ich wynikiem są liczne, często wykorzystywane także i dzisiaj, techniki i narzędzia do opisu i badania własności programów. Najważniejsze z nich przedstawiono w paragrafie 2.2.3.

Jednym z pierwszych badaczy zajmujących się wykorzystaniem metod formalnych w informatyce był McCarthy (1962). On to właśnie stworzył teorię matematyczną służącą do opisu i badania własności programów. Oparł ją na teorii funkcji rekurencyjnych, wzbogaconej o wyrażenia warunkowe; w ten sposób chciał stworzyć dostatecznie silny aparat matematyczny. Dzięki niemu mógł dowodzić, że programy są poprawne w rozumieniu szerszym niż tylko ich zgodność ze specyfikacją (czyli że program „rozwiązuje problem"), oraz dokonywać automatycznych tłumaczeń pomiędzy językami programowania.

Jedną z pierwszych logik służących do dowodzenia poprawności programów stworzył natomiast Hoare (1969). Uważał on, że poprawność działania można wyrazić w terminach związków zachodzących pomiędzy wartościami pewnych zmiennych, przed wykonaniem programu – i po nim; zmienne te formułował w terminach logiki i operacji arytmetycznych. Podane przez niego aksjomaty i reguły dowodowe miały umożliwić dowodzenie poprawności w sposób czysto formalny. Hoare, choć odniósł częściowy sukces, przeprowadzając formalne dowody poprawności dwóch prostych programów, miał świadomość ograniczeń swej metody. Nie można jej było stosować do wszystkich języków programowania, ponieważ operacja podstawiania w nich wykorzystana nie mogła powodować efektów ubocznych. Co więcej, w systemie tym nie jest możliwe przeprowadzanie dowodu, że program zakończy swoje działanie. I, co najistotniejsze, Hoare rozumiał, że poprawność działającego programu zależy od czegoś więcej niż tylko od weryfikacji formalnej – w tym, między innymi, od sprzętu, a w szczególności od sposobu reprezentacji liczb; stworzył on nawet aksjomatykę dla „arytmetyki komputerowej".

Przykładem wykorzystywanych do dziś technik, powstałych w wyniku badań nad formalną weryfikacją programów, jest metoda niezmienników

Floyda, zapoczątkowana przez Naura (1966). Twierdził on, że aby móc mówić o programach jako o obiektach statycznych oraz dynamicznych jednocześnie, należy zatrzymywać ich wykonanie od czasu do czasu i opisywać stany maszyny w tych momentach. Jeśli w programie umieści się dostateczną liczbę punktów kontrolnych (to jest miejsc zatrzymania), można wówczas badać związki, jakie – wyrażone za pomocą formuł – zachodzą pomiędzy stanami maszyny w tych właśnie punktach. Wywodząca się z tej idei metoda niezmienników Floyda polega na umieszczeniu takich punktów na diagramach sekwencji działań programów (a właściwie algorytmów) i na sprawdzeniu pewnych warunków nazywanych niezmiennikami. Powstałe w ten sposób dowody częściowej poprawności programów są typowymi dowodami matematycznymi, w których zazwyczaj wykorzystuje się indukcję. Niektórzy badacze twierdzą jednak, że metoda ta nie daje stuprocentowej gwarancji co do poprawnego wykonania programu, gdyż zawarte w niej rozumowania są podważalne, wykorzystują bowiem założenia dotyczące typowości nieobserwowalnych przypadków (między innymi założenie o poprawnym funkcjonowaniu wykorzystywanego oprogramowania i sprzętu).

Najczęściej dyskutowanym argumentem na rzecz paradygmatu matematycznego, związanym z praktyczną realizacją formalnej weryfikacji programów, jest argument ze zwiększania skali. Wyraża się go zazwyczaj poprzez stwierdzenie, że weryfikacja dużych programów jest sumą wielu małych weryfikacji jego składowych – z czego wynikałoby, że z czasem powstałyby formalne weryfikatory dla rzeczywistych, nawet bardzo skomplikowanych programów. Argument ten jest szeroko krytykowany. Najwięcej wątpliwości budzi twierdzenie, że duży system nie jest niczym więcej niż tylko sumą swoich składowych. Nie powinno się również utożsamiać algorytmów i programów – wystarczy porównać ich specyfikacje, by stwierdzić, że są to inne obiekty, a rozwój jednych nigdy nie doprowadzi wprost do drugich. Co więcej, nie zawsze można przewidzieć działanie programów w sposób czysto analityczny, często sami autorzy są zaskoczeni sposobem ich funkcjonowania. Ponadto dowody formalne w matematyce są często długie i trudne do zrozumienia – i podobnie jest w przypadku formalnych weryfikacji rzeczywistych programów. DeMillo, Lipton i Perlis (1979) twierdzą, że weryfikacje programów – pomimo tego, że są ciągami formuł logicznych – nie są dowodami w sensie matematycznym, ponieważ ich akceptacji nie towarzyszą procesy społeczne charakterystyczne dla uznawania wyników matematycznych.

Zwolennicy metod matematycznych w informatyce przyjmują najczęściej, że pojęcie poprawności programu sprowadza się do jego zgodności z formalną specyfikacją. Jednak sama idea tworzenia takich specyfikacji jest bardzo dyskusyjna. Jak pokazują analizy zawarte w paragrafach 2.1.5 oraz

2.2.3, nie ma powszechnej zgody ani co do tego, jakie specyfikacje powinny być tworzone, ani nawet do tego, jakie są obecnie wykorzystywane (w praktyce). Jeśli jednak, wbrew licznym zastrzeżeniom, uzna się potrzebę tworzenia formalnych specyfikacji, to nadal pozostaje trudność pokazania jej zgodności z programem. Twierdzi się, że tak rozumiana weryfikacja programów ma sens jedynie przy założeniu, że i program, i jego specyfikacja powstają niezależnie, a także – że nie ulegają one zmianom. Wiadomo jednak, iż podczas procesu tworzenia oprogramowania wzajemne oddziaływania pomiędzy nimi są nader częste, tak więc w efekcie wprowadza się zmiany zarówno w specyfikacji, jak i w samym programie.

Warto zwrócić uwagę, że głównym powodem w dążeniu do formalizacji programowania jest twierdzenie, że jest ona niezbędna w dowodzeniu poprawności programów, która to poprawność stanowi absolutny priorytet dla informatyków. Twierdzenie to można jednak łatwo obalić. Pewne własności programów są bowiem do udowodnienia niezależnie od tego, czy zostały one sformułowane w sposób formalny, czy też nieformalny. Przykłady takich rozumowań podają między innymi Aho, Hopcroft i Ullman (1974). Co więcej, wiele powszechnie używanych programów zawiera błędy, a to już dowodzi, że poprawność programu nie jest priorytetem programowania.

Ważną kwestią filozoficzną związaną z programowaniem jest rozróżnienie pomiędzy modelami matematycznymi a ich fizycznymi odpowiednikami. W paragrafie 2.2.3 przedstawiono wybrane poglądy dotyczące tej kwestii. Fetzer (1991) twierdzi, popierając swoją opinię przykładami, że prawdziwe zdania matematyki mogą okazać się empirycznie fałszywe. Zatem nie jest możliwe stworzenie matematycznego dowodu dotyczącego własności programu wykonanego na konkretnym obiekcie fizycznym (komputerze), ponieważ nie istnieją prawdy matematyczne dotyczące obiektów fizycznych. Barwise (1989) natomiast dzięki analizie wykorzystania matematyki do badania rzeczywistości fizycznej zauważa, że jeśli w ogóle powinno się mówić o programowaniu jako części matematyki, to powinna to być matematyka stosowana (a nie matematyka czysta), która co prawda nie gwarantuje wiedzy pewnej, dotyczącej rzeczywistości fizycznej, ale może prowadzić do uzasadnienia pewnych poprawnych przypuszczeń z nią związanych.

Przedstawione w podrozdziale 2.2 analizy pokazują, że autorzy piszący o metodach tworzenia programów, a w szczególności zwolennicy paradygmatu matematycznego, bardzo często nie odróżniają modelu matematycznego od modelowanych obiektów. Należy pamiętać, że formalne weryfikacje, jako dowody matematyki stosowanej, dotyczą modelu – a nie działania programu na fizycznym urządzeniu (komputerze). Ponadto w przypadku

weryfikacji systemów komputerowych (to znaczy programów i urządzeń, na których są one wykonywane) powinno się stworzyć matematyczne modele otoczenia, w jakim one działają i którego częścią są użytkownicy. Wydaje się, że stworzenie takich właśnie modeli jest krańcowo trudne, jeśli nie niemożliwe. Jeżeli nawet modele takie powstaną, to problematyczne będzie stwierdzić, czy są one adekwatne dla danego fragmentu rzeczywistości. Dlatego też wielu informatyków opowiada się za przyjęciem w procesie tworzenia oprogramowania wszystkich skutecznych metod – czyli nieograniczaniem się do metod formalnych.

Praktyka programowania, na którą zwracają uwagę między innymi Scherlis, Hamming i Parnas, wskazuje, że należy oddzielić jakość programów (jako rezultatów procesu programowania) od metod wykorzystywanych podczas ich tworzenia. Scherlis (1989) twierdzi, że programista powinien móc używać wszelkich metod, które prowadzą go do celu, jakim jest stworzenie dobrego programu, dowód jego poprawności natomiast należy rozumieć jako rodzaj dokumentacji. Hamming (1969) zauważa z kolei, że idea dowodzenia poprawności programu przed jego wykonaniem najczęściej nie znajduje zastosowania w praktyce. Większość istniejących programów działa zazwyczaj poprawnie bez wykorzystania w procesie ich tworzenia metod formalnych, bowiem metody inżynieryjne wystarczają do zbudowania dość wiarygodnych systemów w rozsądnym czasie i po rozsądnych kosztach. Parnas (1989) twierdzi natomiast, że testowanie i uzasadnienie matematyczne dopełniają się i że oba te procesy są potrzebne. Wydaje się zatem, że programowanie nie jest obecnie działalnością matematyczną, co oczywiście nie wyklucza, że taką się stanie.

Rozdział 3

Informacja

„Informacja" to termin bardzo stary, który zyskał nowe znaczenie wraz z pojawieniem się komputerów. Informacja stała się bowiem przedmiotem zainteresowań nie tylko filozofów, ale i informatyków[1].

W wielu, o ile nie w większości, pojawiających się obecnie definicjach informatyki spotyka się twierdzenie, że jest ona nauką zajmującą się badaniem różnych aspektów informacji. Na przykład, w *Encyklopedii PWN* pod hasłem „informatyka" znajdujemy określenie: „dyscyplina naukowa zajmująca się przetwarzaniem informacji z użyciem komputerów"[2]. Twierdzi się (por. np. Gorn, 1963; Forsythe, 1967; Atchison i in., 1968; Denning, 1981, 1985, 1989, 1995; Denning i in., 1981, 1989; Hartmanis i Lin, 1992; Hartmanis, 1993), że informatycy zajmują się badaniem sposobów reprezentowania i przetwarzania informacji w systemach informatycznych, efektywnością tych procesów, możliwościami ich zastosowań, jak również naturą samej informacji – jej filozofią. Hartmanis i Lin (1992) piszą: „Informatycy koncentrują się na informacji, na sposobach jej reprezentowania i przetwarzania, na maszynach oraz systemach, które realizują te zadania" (s. 164). Z kolei Forsythe (1967) twierdzi, że „[i]nformatyka w ogólności jest sztuką i nauką reprezentowania i przetwarzania informacji, a w szczególności przetwarzania jej za pomocą maszyn logicznych nazywanych automatycznymi komputerami cyfrowymi" (s. 4).

---

[1] Warto w tym miejscu podkreślić, że całkowicie poprawne i uzasadnione jest mówienie o informacji w kontekście prac autorów żyjący przed dobą komputeryzacji. Analiza dzieł takich twórców jak Platon, Kartezjusz czy Nitzsche może być bardzo owocna, szczególnie w dziedzinie metodologii informacji.

[2] http://encyklopedia.pwn.pl/haslo/3914698/informatyka.html

Warto zwrócić uwagę, że zazwyczaj pod pojęciem informacji rozumie się zarówno to, co przetwarzają komputery, jak i informację naturalną – wpisaną w otaczającą nas rzeczywistość przyrodniczą. Zatem można stwierdzić, że informatyka bada nie tylko urządzenia i techniki przetwarzania informacji (w szczególności metody jej efektywnej transmisji), ale również informację zawartą na przykład w kodzie genetycznym.

Rozdział ten poświęcimy analizie filozoficznej szeroko rozumianego pojęcia „informacja", rozpoczynając od omówienia różnych sposobów jej definiowania.

3.1. Definicje informacji

W czasach współczesnych pytanie o definicję i naturę informacji jest jednym z podstawowych i najczęściej zadawanych w filozofii nauki. Pojęcie „informacja" pojawia się bowiem w wielu – jeśli nie w większości – dyscyplinach naukowych, w szczególności w biologii, fizyce, naukach społecznych i ekonomicznych. Czy w kontekstach różnych nauk pojęcie to ma takie samo znaczenie? I w szczególności, czym jest informacja w informatyce?

Wbrew powszechnemu przekonaniu źródeł pojęcia „informacja" należy szukać w historii starożytnej, a nie w dwudziestowiecznych rozważaniach dotyczących komputerów jako urządzeń przetwarzających i przesyłających informacje.

Termin „informacja" pochodzi od łacińskiego słowa *informatio*, które ma dwa znaczenia: 1) proces nadawania formy czemuś materialnemu oraz 2) akt komunikowania wiedzy innej osobie (Capurro, 2009). Po raz pierwszy pojawia się on w pracach starożytnych filozofów: w platońskiej teorii idei – wiecznych bytów niedostępnych poznaniu zmysłowemu – oraz u Arystotelesa, w pojęciu formy jako tego, co nadaje kształt i postać materii. Omawiając teorie starożytnych Greków, Cyceron i św. Augustyn używają łacińskich słów *informare* oraz *informatio* dla oznaczenia takich terminów jak: idea (*idea*), esencja (*eidos*) czy forma (*morphe*). Warto zauważyć, że rdzeń „forma" jest obecny w wielu językach w słowach określających informację: w angielskim in-*form*-ation, a w języku polskim in-*forma*-cja.

Podobnie wyglądają dzieje początków matematycznych teorii opisujących informację; początków ich nie należy szukać w teorii XX-wiecznej komunikacji Shannona, lecz już w starożytności. Wtedy bowiem opisywano formę bytu jako wzorzec lub strukturę, którą można przedstawić za pomocą liczb. Opisy takie miały dwa aspekty: ontologiczny (wyjaśniały istotę rze-

czy) i epistemologiczny (przedstawiały sposoby jej poznania). Można zatem stwierdzić, że pojęcie „informacja" ma od początku związek z ontologią, epistemologią i matematyką.

Historia pojęcia „informacja" w nowożytnej filozofii jest bardzo skomplikowana. Zniknęło ono bowiem prawie całkowicie z analiz filozoficznych, zyskało natomiast popularność w mowie potocznej. Około XIV wieku termin ten pojawił się w wielu językach europejskich w kontekście nauczania. Jest to jedno z trzech różnych, istotnych historycznie, znaczeń tego słowa.

Najstarsze z nich, związane również z edukacją, to rozumienie „informacji" jako procesu, mianowicie – bycia informowanym. W filozofii klasycznej mówiono o rozpoznaniu obiektu jako na przykład konia, gdy „forma" konia „zakorzenia się" w umyśle – była to właśnie „informacja" o naturze tego zwierzęcia. Takie pojmowanie „informacji" nakierowane jest na proces. Z jego rezultatem związane jest drugie ze znaczeń informacji – jako stanu agenta. Jeśli nauczamy w szkole twierdzenia Pitagorasa, to po zakończeniu procesu jego poznawania przez uczniów możemy powiedzieć, że mają oni informacje na jego temat. Co więcej, jeśli zapoznając uczniów z tym zagadnieniem, korzystamy z jego przedstawienia w podręczniku, to naturalne jest powiedzieć, że podręcznik zawiera informacje o twierdzeniu Pitagorasa. W ten sposób dochodzimy do trzeciego znaczenia terminu „informacja", czyli zdolności obiektu (książki) do informowania agenta (ucznia). W tym samym sensie mówimy o informacji w sytuacji, gdy uczeń po otrzymaniu jej od nauczyciela przekazuje ją swojemu koledze. Informacja staje się wówczas czymś, co może być przechowywane, przekazywane i mierzone.

W języku potocznym rzeczownik „informacja" jest używany w głównej mierze dla oznaczenia jakiejś ilości danych, kodu lub tekstu, które są przechowywane, przesyłane lub przetwarzane w dowolnym medium. Uniwersalność tego terminu – związana z jego powszechnym użyciem – idzie w parze z brakiem precyzji w jego definiowaniu.

Próbę sprecyzowania potocznego rozumienia pojęcia „informacja" podjął Jacek J. Jadacki (2003). Proponuje on następujące określenie:

> Jeżeli *x* wygłasza w obecności *y*-a w chwili *t* zdanie *Z*, to *x* przekazuje *y*-owi w chwili *t* **informację** o tym, czy *p*, gdy:
> (a) *y* w okresie (bezpośrednio) przed chwilą *t* nie wie, czy *p* (w szczególności z wiedzy *y*-a z okresu przed *t* nie wynika ani to, że *p*, ani to, że nie-*p*);
> (b) *y* w chwili *t* odbiera (*scil*. słyszy) zdanie *Z*:
> – rozumiejąc zdanie *Z*;
> – uznając zdanie *Z* za dostatecznie uzasadnione i prawdziwe;
> [i tym samym:]
> (c) *y* w okresie (bezpośrednio) po chwili *t* wie, czy *p*. (s. 8, wyróżnienie oryginalne).

Istniejące definicje informacji można podzielić na dwie grupy: ogólne (zunifikowane) – odpowiednie dla wszystkich zastosowań tego terminu, oraz specyficzne – dla poszczególnych nauk. Definicje należące do pierwszej grupy starają się ująć cechy informacji wspólne wszystkim dziedzinom, w których występuje to pojęcie (również odnoszące się do potocznego jego rozumienia), natomiast te należące do drugiej ograniczają się do konkretnej nauki – czym innym jest informacja w biologii, czym innym zaś w socjologii czy informatyce. Można jednak przyjąć jeszcze inną klasyfikację.

Nawrocki (2003) zaproponował podział istniejących definicji informacji na trzy grupy, biorąc pod uwagę kryterium przedmiotowe, to znaczy zjawiska i procesy, na których poszczególne definicje się koncentrują.

Pierwszą grupę stanowią definicje reprezentujące podejście „strukturalistyczne", zakładające istnienie w każdym obiekcie materialno-energetycznym informacji o nim samym – a jest ona niezależna od zewnętrznego obserwatora. Informacja ta „zakodowana" jest jako konstruktywny, dynamiczny ład wewnętrzny jego makro- i mikrostanów, z regułami oddziaływań, transformacji i utrzymywania koherencji. Informacja stanowi wtedy trzeci, poza materią i energią, składnik wszelkiego bytu realnego, wyrażający jego wewnętrzną organizację.

Definicje grupy drugiej – „interakcyjne", opisują informację jako odzwierciedlenie zmian, które dokonują się w toku wzajemnego oddziaływania różnych obiektów realnych, poprzez nośniki sygnałów i za pośrednictwem wspólnych środowisk. Każde takie oddziaływanie polega na wzajemnych reakcjach powodujących zmiany w obiektach oddziałujących, a odbiór informacji warunkowany jest możliwością ich rozpoznania i absorpcji.

Cechą charakterystyczną definicji grupy trzeciej – „komunikatywistycznych", jest przyjmowanie założenia, że informacja nie istnieje bez przetwarzającego ją odbiorcy. Istnienie jej ogranicza się zatem do sytuacji, w której mamy do czynienia z sygnałem niosącym treść aktywnie przetwarzającemu go odbiorcy. Nawrocki (2003) pisze: „Informacja jest odwzorowaniem odebranego sygnału zmiany (inności, różnicy) przez układ recepcyjno-przetwarzający odbiorcy i rezultatem porównywania treści tej zmiany z jego zasobami informacyjno-znaczeniowymi" (s. 53).

Nawrocki (2003) proponuje również własną definicję informacji jako „treści sygnału mającej sens dla odbiorcy" (s. 56). Wyjaśnia przy tym, że sygnał to zbiór zmian odpowiadających danemu schematowi, treść to skutek aktu poznania („wypełnienie" pojęcia), natomiast sens jest logicznym odniesieniem do zbioru znaczeń. „Posiadanie sensu" odbiorca rozumie jako zależność polegającą na tym, że informacja kształtowana jest przez sygnał, odbiorcę oraz jego reakcję, polegającą na przypisaniu temu sygnałowi sensu. Informacja ma zatem sens nadany jej przez odbiorcę sygnału. Autor

twierdzi, że jego definicja ma zastosowanie do interpretacji komunikatywistycznej i interakcyjnej, nie odpowiada jednak rozumieniu strukturalistycznemu. Jego zdaniem, w definicjach strukturalistycznych stosowanie pojęcia „informacja" jest nadużyciem, które można usprawiedliwić jedynie brakiem innego, adekwatnego terminu. Nie można bowiem mówić o informacji bez relacji do jej zewnętrznego odbiorcy.

Jeszcze inną klasyfikację definicji informacji zaproponował Floridi (2004c), dzieląc je na redukcjonistyczne, nieredukcjonistyczne oraz antyredukcjonistyczne[3].

Twórcy definicji redukcjonistycznych próbują stworzyć ogólną teorię, twierdząc, że wszystkie rodzaje informacji są redukowalne do pewnych pojęć bazowych (ang. *ur-concepts*). Warto przy tym zauważyć, że dotychczas – pomimo rozwoju wielu takich teorii – nie udało się wypracować jednej, powszechnie akceptowanej zunifikowanej definicji informacji.

Przeciwne stanowisko prezentują antyredukcjoniści – podkreślający różnoraką naturę informacji oraz pojęć z nią związanych. Uważają oni, że informacja ma w różnych dyscyplinach naukowych nie tylko odmienne zastosowanie, ale nawet pochodzenie. Zatem próby stworzenia jednej teorii informacji dla wszystkich nauk są z góry skazane na niepowodzenie i należy rozwijać różne, niezależne od siebie koncepcje.

Nieredukcjoniści natomiast chcą uniknąć dychotomii pomiędzy koncepcjami redukcjonistycznymi i antyredukcjonistycznymi – na przykład poprzez zastąpienie jednego pojęcia bazowego siecią wielu pojęć centralnych, które mogą (ale nie muszą) być ze sobą powiązane. Zgodnie z takimi koncepcjami nie ma jednego kluczowego pojęcia informacji; jest ona widziana bardzo różnorodnie, jako interpretacja, zdolność (ang. *power*), relacja, wiadomość lub przekaz, rozmowa, konstrukcja, a nawet – towar[4].

Najwcześniejsze i jedno z najbardziej popularnych sformułowań ogólnej definicji pochodzi od Batesona (1973): „W rzeczywistości to, co rozumiemy przez informację – podstawową jej jednostkę – to różnica, która czyni różnicę (ang. *a difference which makes a difference*)" (s. 428).

Rozmaite interpretacje występującego w tej definicji pojęcia „różnicy" doprowadziły do powstania wielu różnorodnych określeń informacji. Na przykład Floridi (2004a) – tworząc swoją ogólną definicję informacji – twierdzi[5], że owa „różnica" jest stanem dyskretnym (a więc daną), nato-

---

[3] „Czym jest informacja?" jest jednym z pytań nazywanych sokratejskimi – „co to jest x?". Zazwyczaj odpowiedzi na takie pytania można podzielić na takie właśnie trzy duże grupy.

[4] Tacy filozofowie jak Baudrillard, Foucault, Lotard, McLuhan, Rorty czy Derrida, których poglądy różnią się w szczegółach, są zgodni co do jednego: informacja nie jest *w*, *z* ani *o* rzeczywistości. Bagatelizują oni również fakt, że informacja czegoś dotyczy (mówi o czymś).

[5] Sformułowanie tej definicji oraz jej analizę znaleźć można w paragrafie 3.2.5.

miast „czynienie różnicy" oznacza, że dana ta jest, przynajmniej potencjalnie, znacząca.

Próbę sformułowania ogólnej definicji informacji, odpowiedniej dla wszystkich nauk posługujących się tym terminem, podjął również George Ifrah (2006):

> Informacja stanowi zespolenie elementu formującego i elementu formowanego, a istotę tego zespolenia określa zmiana stanu polegająca na tym, że pojawienie się elementu formującego pobudza element formowany zgodnie z regułami ustalonego z góry kodu[6] (s. 868).

Określenie to jest, co prawda, bardzo ogólne, ale jako takie wydaje się niczego nie wyjaśniać. Co więcej, nie odnosi się ono do kontekstu, w którym najczęściej pojawia się termin „informacja", a mianowicie do zjawiska komunikacji.

Dotychczasowe próby stworzenia ogólnej definicji informacji pokazują, że jest to zadanie bardzo trudne, o ile nie niemożliwe. Claude Shannon, twórca matematycznej teorii komunikacji, pisał (1953): „Trudno spodziewać się, że jedno (pojedyncze) pojęcie informacji może w satysfakcjonujący sposób służyć licznym możliwym zastosowaniom" (s. 180). Trudności z ogólną definicją informacji, odpowiednią dla wszystkich nauk, dobrze oddaje *Encyklopedia PWN*: „Informacja [łac. informatio 'wyobrażenie', 'wyjaśnienie', 'zawiadomienie'], pojęcie (*w zasadzie niedefiniowalne*) występujące w teorii informacji"[7].

Obecnie coraz popularniejszy staje się pogląd, że liczne pojęcia informacji, występujące w różnych naukach, są bardziej lub mniej izolowane, a stworzenie ogólnej jej teorii wydaje się niemożliwe (por. np. Adriaans, 2013). Znacznie łatwiejsze okazuje się definiowanie informacji w obrębie danej dyscypliny naukowej.

Przykładem takiego „lokalnego" podejścia do definiowania informacji jest jej powszechne w informatyce rozumienie jako „sygnałów lub symboli, które niosą znaczenie dla ludzkiego obserwatora" (Denning, 1999, s. 25). Informacja przechowywana w komputerze jest, w czysto technicznym sensie, ciągiem bitów (zer i jedynek). Jednakże w sensie intuicyjnym/praktycznym informacja zakodowana w tych bitach jest pojęciem pochodnym, zależnym od kodowania (kodującego), programu (programisty) oraz od interpretującego ją (użytkownika).

Niektórzy badacze twierdzą, że informacja jest założeniem czynionym przez każdego obserwatora, ale nie istnieją powszechnie akceptowane

---

[6] George Ifrah twierdzi, że jego rozumienie informacji jest całkowicie zgodne z opisaną poniżej ilościową teorią informacji Shannona, ponieważ element formujący jest tym (jedynym!) elementem, który poddaje się obserwacji i pomiarowi ilościowemu.

[7] http://encyklopedia.pwn.pl/haslo.php?id=3914686. Wyróżnienie moje [I. B-K].

standardy tych założeń. Takie rozumienie jej jest obecnie silnie krytykowane. Skrajnie nienaukowe wydaje się bowiem twierdzenie, że informatyka jako nauka bazuje na pojęciu, które jest fundamentalnie subiektywne. Twierdzi się nawet (por. Talbott, 1999), że przyjęcie poglądu, iż informacja jest obiektem badań informatyki, prowadzi do problemów związanych z jakością tworzonego przez informatyków oprogramowania oraz do trudności w edukacji młodych adeptów tej nauki.

3.2. Wybrane teorie informacji

Jeśli przyjąć, że przedmiotem dociekań informatyków są szeroko rozumiane zagadnienia związane z informacją, to nie można poprzestać jedynie na zdefiniowaniu jej (jako pojęcia), konieczna jest również analiza jej natury. Jakie cechy ma informacja?

Powstało już – i nadal powstaje – wiele teorii informacji, a wybrane z nich przedstawiamy w tym rozdziale. Niektóre koncentrują się tylko na pewnych aspektach informacji, inne próbują uchwycić pełną jej charakterystykę. Rozpocznijmy od analizy – najbliższych informatyce – teorii ilościowych.

3.2.1. Matematyczna teoria komunikacji

Jednymi z podstawowych zagadnień związanych z informacją, którymi zajmuje się informatyka, są przesyłanie informacji i jej mierzenie. Dlatego też termin „teoria informacji" jest przez większość informatyków utożsamiany z matematyczną teorią komunikacji, koncentrującą się właśnie na tych zagadnieniach,

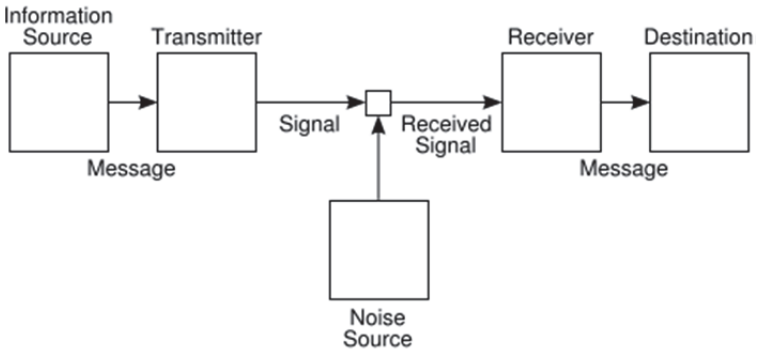
Istnieje wiele sposobów mierzenia „ilości informacji", jednak większość z nich stara się oddać dwie podstawowe intuicje. Po pierwsze, dłuższy komunikat (na przykład dłuższy tekst) zawiera potencjalnie więcej informacji – to znaczy, że informacja jest ekstensywna. Po drugie, zmniejsza ona niepewność – jeśli jesteśmy całkowicie pewni co do stanu rzeczy, to nie możemy otrzymać o nim żadnej więcej informacji. A to z kolei sugeruje związek pomiędzy informacją a prawdopodobieństwem – mało prawdopodobne komunikaty zawierają więcej informacji[8]. Tę właśnie korelację wykorzystali

[8] W codziennym życiu największe poruszenie wywołują wiadomości mało prawdopodobne, zaskakujące, zawierające dużą ilość informacji. Jeśli usłyszymy w radiu komunikat o tym, że „jutro wzejdzie słońce", uznamy go za banał, a nie za informację. Komunikat o tym, że klasowy prymus napisał egzamin na ocenę bdb, nie wywoła u nauczyciela większego zainteresowania, ale stwierdzenie, że najsłabszy uczeń otrzymał ocenę bdb, to już jest

twórcy matematycznej teorii komunikacji Claude E. Shannon oraz Warren Weaver, przedstawiając ją w sposób najbardziej kompletny w pracy z 1948 roku *The Mathematical Theory of Communication* – na niej to w dużej mierze oparty został poniższy opis[9].

Matematyczna teoria komunikacji opisuje w sposób formalny zapis (kodowanie), przesyłanie i dekodowanie informacji. Zadaniem jej jest jednocześnie pogodzenie dwóch przeciwstawnych celów: jak najzwięźlejsze zapisywanie wiadomości oraz chronienie jej przed przekłamaniami podczas transmisji. Teoria ta traktuje informację jak zjawisko fizyczne związane z wybieraniem jednego symbolu ze zbioru możliwych, przy czym największą ilość informacji niesie tekst, w którym prawdopodobieństwo wystąpienia każdej litery jest takie samo, to znaczy ciąg losowy. Stworzona przez Shannona i Weavera teoria opisuje informację w sposób czysto syntaktyczny. Używany w niej termin „informacja" ma wyłącznie znaczenie techniczne; całkowicie pomija się nie tylko jej znaczenie, ale również kontekst. Jest to zatem czysto ilościowa teoria informacji – opisująca tylko jeden z jej aspektów, a nie teoria opisująca informację w jej potocznym rozumieniu, choć za taką uważa ją wielu filozofów. Nawet jej twórcy zaprzeczali związkom ich pojęcia „informacja" z tym samym słowem używanym w życiu codziennym.

Na szczególną uwagę zasługuje przyjęty przez Shannona model komunikacji, ponieważ jest on wykorzystywany przez wielu twórców teorii informacji – w tym tych teorii, które nie są ograniczone do kwestii wyłącznie ilościowych i syntaktycznych. Model ten najczęściej przedstawiany jest za pomocą liniowego diagramu (rysunek 3).



Rys. 3. Schemat ogólnego modelu komunikacji, źródło: Shannon, 1948, s. 381

informacja! Co więcej, ta informacja w przypadku wielu nauczycieli zmusi ich do działania, na przykład do poszukiwania osoby, od której uczeń ten „ściągnął".

[9] Shannon prawdopodobnie po raz pierwszy użył terminu „teoria informacji" w 1945 roku w pracy zatytułowanej *A Mathematical Theory of Cryptography*.

W powyższym modelu istotne jest założenie, że informujący (znajdujący się na początku diagramu) oraz informowany (na jego końcu) posiadają taką samą wiedzę bazową dotyczącą zbioru używanych symboli, czyli alfabetu. Wyróżnia się w nim następujące elementy:

- źródło informacji (informator) – to coś lub ktoś generujące/y wiadomość,
- wiadomość – ciąg znaków (w dowolnej postaci, na przykład słownej, znakowej czy obrazkowej) generowanych przez źródło,
- nadajnik – urządzenie kodujące wiadomość (ang. *transmitter*) w postaci odpowiedniej do jej przesłania,
- odbiornik dekodujący wiadomość (ang. *receiver*), pełniący funkcję odwrotną do nadajnika,
- punkt docelowy informacji – urządzenie lub osoba, która jest odbiorcą komunikatu,
- źródło zakłóceń – proces lub zdarzenie pojawiające się podczas transmisji, będące przyczyną powstania szumu, oraz
- kanał komunikacyjny – miejsce przesyłania komunikatu, charakteryzujące się wieloma parametrami, w tym przepustowością (lub pojemnością) zależną od jego struktury fizycznej.

Warto zauważyć, że Shannon i Weaver zakładali pewną uniwersalność swego modelu, nie ograniczając go tylko do przesyłania sygnałów za pomocą urządzeń technicznych. Weaver (1949) podaje następujący przykład: „Gdy do ciebie mówię, mój mózg jest źródłem informacji, a twój – punktem docelowym; mój układ wokalny jest przekaźnikiem, a twoje ucho wraz z ośmioma nerwami jest odbiornikiem" (s. 29).

Aby zrozumieć, czym jest informacja opisywana przez matematyczną teorię komunikacji, należy przyjrzeć się metodzie jej pomiaru. Ilość informacji, jaka związana jest z danym komunikatem, zależy od stopnia jej nieprzewidywalności: im mniej prawdopodobny jest ów komunikat, tym więcej informacji ze sobą niesie.

Spróbujmy wyobrazić sobie urządzenie, które generuje tylko jeden rodzaj informacji – tak zwane źródło unarne. Przykładem jego może być kruk[10], który na wszystkie pytania odpowiada zawsze tak samo: „nigdy więcej". Pełni on wówczas rolę informującego, słuchacze są informowanymi, a „nigdy więcej" to komunikat (wiadomość). Unarne źródła informacji, takie jak ów kruk, produkują zerową ilość informacji, gdyż niezależnie od zadawanego pytania odpowiedź jest całkowicie przewidywalna – zawsze taka sama (wyraża się tym samym komunikatem)[11].

---

[10] Ptak ten nazywany jest krukiem Poego (por. np. Floridi, 2004c).

[11] Milczące źródło jest również źródłem unarnym i produkuje zerową informację.

W przypadku źródeł binarnych, takich jak na przykład symetryczna moneta, lub innych źródeł generujących dwa różne symbole sytuacja wygląda następująco. Przed podaniem wyniku rzutu monetą (lub też po prostu przed nadaniem jednego z dwóch możliwych symboli) „niewiedza" informowanego (jego deficyt danych) jest niezerowa, ponieważ nie wie on, który symbol „wyprodukuje" urządzenie (nie wie na przykład, czy wypadł orzeł, czy reszka). Taki deficyt danych określa się mianem „niepewności". Zauważmy, że może to być termin mylący, gdyż nasuwa on pewne skojarzenia semantyczne, które z omawianą teorią nie mają nic wspólnego. Rzut monetą produkuje informację, która jest funkcją możliwych wyników (w tym przypadku dwóch jednakowo prawdopodobnych symboli) i jest równa deficytowi danych, jaki usuwa.

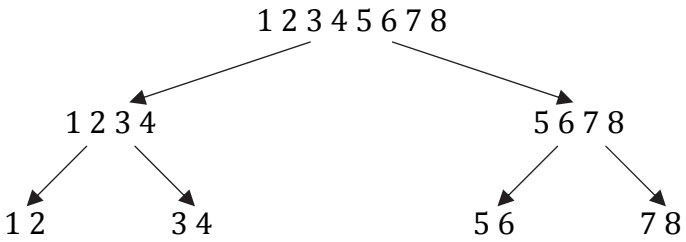
Sytuacja komplikuje się w przypadku bardziej złożonych źródeł informacji. Rozważmy dla przykładu układ złożony z dwóch symetrycznych monet: A i B. Taki system może wygenerować cztery różne wyniki opisane przez pary uporządkowane: <O,O>, <O,R>, <R,O>, <R,R>, gdzie R oznacza, że na monecie wypadła reszka, natomiast O – że wypadł orzeł. Na pierwszym miejscu w każdej parze znajduje się wynik z monety A, a na drugim – z monety B. Układ ten generuje deficyt danych równy 4 jednostki, ponieważ tyle jest możliwych wyników. Dodając jeszcze jedną monetę, otrzymamy układ o deficycie danych równym 8 jednostek – i tak dalej….

Uogólniając powyższe rozważania, otrzymujemy następujący związek. Niech N oznacza liczbę możliwych komunikatów, dla $N = 1$ ilość informacji generowanej przez źródło wynosi 0 (źródło unarne), z kolei źródło binarne (gdy $N = 2$) – produkuje 1 jednostkę informacji, w przypadku gdy $N = 4$ (na przykład w przypadku rzutu dwiema monetami), ilość informacji generowanej przez źródło można obliczyć, wykorzystując ilości informacji generowanej przez monetę A oraz przez monetę B. Miara informacji jest zatem funkcją prawdopodobieństwa wystąpienia odpowiednich symboli. Shannon i Weaver do jej obliczenia użyli logarytmu o podstawie 2, jest ona odpowiednia ze względu na używanie sprzętu cyfrowego (w tym komputerów) do reprezentacji danych. Jednostką informacji jest wówczas bit (od ang. ***bi****nary digi****t***). Wybór takiej podstawy logarytmu jest jednak tylko kwestią konwencji. Jeśli przyjmiemy, że jest nią liczba e – jednostka informacji nazywa się nat (nit), natomiast dla podstawy 10 – dit lub hartley. Logarytmy mają dodatkowo tę użyteczną własność, że zamieniają mnożenie (liczby symboli) na dodawanie (jednostek informacji). Zatem mając dany alfabet o N jednakowo prawdopodobnych symbolach, możemy obliczyć ilość informacji generowanej przez źródło, korzystając ze wzoru: $\log_2(N)$.

Zastanówmy się teraz, jaka intuicja kryje się za tymi obliczeniami, oraz spróbujmy odpowiedzieć na pytanie, czy istnieje prosty sposób określenia

ilości informacji generowanych przez dowolne źródło. Rozważmy następującą zabawę. Jeden z uczestników wybiera liczbę z pewnego ustalonego zbioru, na przykład spośród liczb 1, 2, …, 8 (jest to odpowiednik nadania jednego komunikatu spośród kilku możliwych). Pozostali uczestnicy próbują zgadnąć, jaka to liczba, zadając pytania, na które mogą otrzymać tylko odpowiedź „tak" lub „nie". Zwycięża ten, kto zgadnie liczbę po zadaniu najmniejszej liczby pytań. Ile, w ogólności, trzeba zadać pytań, by odgadnąć tę liczbę?

Jeden ze sposobów wykonania tego zadania polega na dzieleniu zbioru na dwie części (o możliwie takiej samej liczbie elementów), potem każdej z nich ponownie na dwie części i tak dalej, aż do otrzymania pary liczb, w której jedna jest liczbą szukaną. W przypadku zbioru liczb 1, 2, …, 8 podział może wyglądać następująco:



Odgadnięcie każdej z liczb tego zbioru wymaga zatem trzech pytań, odpowiadających kolejnym podziałom:

1. Czy szukana liczba jest mniejsza od 5?
2. W zależności od odpowiedzi na pytanie 1:
   a. jeśli odpowiedź na pytanie 1 brzmi „tak", to pytanie drugie brzmi: „czy szukana liczba jest mniejsza od 3?"
   b. jeśli odpowiedź na pytanie 1 brzmi „nie", to pytanie drugie brzmi: „czy szukana liczba jest mniejsza od 7?"
3. Ostatnie pytanie to zawsze pytanie o jedną z dwóch liczb, co daje jednoznaczną odpowiedź.

Liczba pytań koniecznych do odgadnięcia komunikatu, jaki został nadany przez źródło (czyli liczba kroków podziału), jest równa ilości informacji generowanej przez źródło.

Aby odgadnąć komunikat nadawany przez źródło unarne (kruka), nie musimy zadawać pytań, a więc niesie ono zerową informację. Do odgadnięcia wyniku jednokrotnego rzutu monetą wystarczy jedno pytanie („czy wypadł orzeł?" lub „czy wypadła reszka?"), więc źródło takie produkuje 1 bit informacji. Aby odgadnąć wynik rzutu dwiema monetami, wystarczą 2 py-

tania (o wyniki na poszczególnych monetach), a zatem źródło takie produkuje 2 bity informacji. Z powyższego przykładu wynika, że do odgadnięcia elementu z ośmioelementowego zbioru wystarczą 3 pytania, a zatem źródło złożone z 3 monet (dla którego możliwych jest 8 wyników) generuje 3 bity informacji – i tak dalej.

Ilość informacji mierzona jest tutaj w terminach zmniejszania deficytu informacyjnego, przy czym zależy ona od prawdopodobieństwa pojawienia się każdego z symboli. Im większe prawdopodobieństwo pojawienia się symbolu, tym mniejsza ilość informacji, jaką on niesie (mniejsza jest jego „informatywność"). Kruk nie generował żadnej informacji, ponieważ pojawienie się komunikatu „nigdy więcej" jest pewne – jego prawdopodobieństwo wynosi 1.

W praktyce przesyłane komunikaty nie są zazwyczaj pojedynczymi symbolami, lecz ich ciągami. Aby obliczyć średnią informatywność takiego ciągu, trzeba uwzględnić informatywność każdego jego symbolu. W przypadku N jednakowo prawdopodobnych symboli, prawdopodobieństwo P_i wystąpienia i-tego symbolu dane jest wzorem $P_i = \frac{1}{N}$. A zatem im mniejsze prawdopodobieństwo wystąpienia symbolu, tym większa jego informatywność (oznaczana przez u): $u_i = -\log(P_i)$.

Rozważmy przykład dziesięciu rzutów monetą, których wynik zapisujemy w postaci ciągu dziesięcioelementowego, w którym na i-tym miejscu umieszczamy symbol O, jeśli w i-tym rzucie wypadł orzeł, lub R – w przeciwnym wypadku, na przykład: <O,O,O,R,R,O,R,O,O,R>. Niech S oznacza długość takiego ciągu (w naszym przykładzie S=10). Oczywiście S to liczba wystąpień reszki plus liczba wystąpień orła. W ogólności – dla i symboli (i rzutów) otrzymujemy wzór:

$$S = \sum_{i=1}^{N} S_i$$

gdzie S_i oznacza liczbę wystąpień danego symbolu w rozważanym ciągu. Zatem wzór na średnią informatywność ciągu symboli o długości S ma postać:

$$\frac{\sum_{i=1}^{N} S_i u_i}{\sum_{i=1}^{N} S_i} = \sum_{i=1}^{N} \frac{S_i}{S} u_i.$$

Iloraz $\frac{S_i}{S}$ jest częstotliwością, z którą i-ty symbol pojawia się w S.

I z kolei, jeśli łańcuch *S* jest dowolnie długi, to częstotliwość *i*-tego symbolu opisuje się za pomocą P_i – prawdopodobieństwa jego wystąpienia. Stąd otrzymujemy miarę informatywności ciągu symboli w postaci:

$$\sum_{i=1}^{N} P_i \, u_i = -\sum_{i=1}^{N} P_i \, log \, P_i$$

gdzie P_i to prawdopodobieństwo zajścia zdarzenia *i*.

Powyższe równanie opisuje zatem średnią ilość informacji przypadającej na znak symbolizujący zajście zdarzenia z pewnego zbioru. Opisywana nim wielkość nosi nazwę entropii (i oznaczana jest literą *H*). Z jej określenia wynika, że entropia jest maksymalna, gdy prawdopodobieństwa poszczególnych zdarzeń są takie same – gdy symbole są jednakowo prawdopodobne. Własność ta prowadzi do pewnego paradoksu.

Skoro największą ilość informacji niesie tekst, w którym prawdopodobieństwo wystąpienia każdej litery jest takie samo – ciąg całkowicie losowy – to na przykład dzieła Szekspira zawierają mniej informacji niż tekst złożony z losowo pomieszanych symboli (liter i znaków przestankowych) z tych dzieł. Co więcej, jak zauważył Aldous Huxley, sto małp piszących na maszynie dostatecznie długo napisze w końcu wszystkie prace Szekspira. Ta całkowicie sprzeczna z intuicją własność informacji została nazwana przez Michaela Dunna (2008) „paradoksem małp", ponieważ całkowite nonsensy wypisywane przez większość czasu przez małpy niosą więcej informacji niż napisany przez nie w końcu *Hamlet*.

Skąd bierze się ów paradoks? Można to wyjaśnić w następujący sposób: gdy małpy piszą na przykład sceny z *Hamleta*, łatwiej jest w takim przypadku przewidzieć następną literę niż gdy piszą one kompletne bzdury – kolejne litery są mniej „przewidywalne". A zatem, nonsensy niosą więcej informacji niż słowa Szekspira.

Michael Dunn (2008) twierdzi jednak, że jest to tylko pozorny „paradoks", nie można bowiem twierdzić, że mniejsze jest prawdopodobieństwo tego, iż losowy ciąg znaków stanowi sensowne słowo, niż tego, że jest zlepkiem nic nie znaczących znaków. Jeśli bowiem małpa wypisuje litery całkowicie losowo, to prawdopodobieństwo pojawienia się jakiegoś sensownego słowa, na przykład „Hamlet" jest dokładnie takie samo, jak pojawienia się słowa „lHmaet"[12]. W całkowicie losowych ciągach wzorce pojawiają się zatem tylko w umysłach obserwatorów.

---

[12] Jeśli w kolejnych pięciu rzutach monetą wypadł orzeł, to prawdopodobieństwo ponownego jego wypadnięcia w rzucie szóstym jest takie samo jak prawdopodobieństwo tego, że tym razem (w końcu) wypadnie reszka.

Powróćmy teraz do omówienia matematycznej teorii komunikacji. W przedstawionych powyżej przykładach mamy do czynienia z pewnymi uproszczeniami: prawdopodobieństwo wystąpienia poszczególnych symboli jest stałe (nie zmienia się w czasie) oraz nie zależy ono od symboli wcześniejszych. Jednakże w przypadku rzeczywistych komunikatów, wyrażonych na przykład w języku polskim, prawdopodobieństwo wystąpienia danej litery w ciągu w znaczący sposób zależy od liter ją poprzedzających.

Jeżeli rozumiemy proces generowania komunikatu jako emisję ciągów symboli, to do ich analizy można wykorzystać teorię procesów stochastycznych opracowaną przez A.A. Markowa (1951) i A.N. Kołmogorowa (1965). Gdy prawdopodobieństwo pojawienia się jakiegoś znaku (zdarzenia) jest zależne od symboli (zdarzeń) poprzedzających go, to proces taki nazywa się procesem – lub łańcuchem – Markowa. Dlatego też Shannon i Weaver (1948) definiują ilość informacji jako stochastyczną własność określonego komunikatu (zdarzenia). W tym przypadku interpretacja informacji przypomina interpretację pewnego układu. Weaver (1949) tak pisze o związkach informacji z entropią:

> To, że informacja może być mierzona za pomocą entropii, jest zupełnie naturalne, jeśli pamiętamy, że informacja związana jest z liczbą wolnych (swobodnych) wyborów, jakich dokonujemy podczas konstruowania wiadomości. Zatem możemy powiedzieć o źródle komunikacji, tak jak moglibyśmy powiedzieć o układzie termodynamicznym: 'Ta sytuacja jest wysoce zorganizowana: nie jest charakteryzowana przez wysoki stopień losowości lub wyboru – to znaczy informacja lub entropia jest niska' (s. 31).

Entropię *H* można zatem interpretować nie tylko jako średnią ilość niezinterpretowanej informacji produkowanej przez informującego (źródło) czy też jako odpowiadającą jej średnią ilość deficytu danych, jaki ma odbiorca przed uzyskaniem komunikatu (czyli tego, co Shannon nazywał „niepewnością"), ale również jako możliwości informacyjne samego źródła. Ostatnia z tych interpretacji prowadzi do rozumienia entropii – a tym samym również informacji – jako wielkości fizycznej, takiej jak masa lub energia.

Entropia może być również traktowana jako miara odwracalności i uporządkowania. Dobrze uporządkowana wiadomość zawiera mały stopień entropii[13] lub losowości, a zatem – mało informacji. Im większa potencjalna

---

[13] „Entropia – miara nieokreśloności, chaotyczności, stopnia nieuporządkowania; *fiz.* wielkość fizyczna charakteryzująca stan układu ciał materialnych i określająca kierunek samoistnych zmian w odosobnionym układzie termodynamicznym (prawo wzrostu entropii); w teorii informacji – miara nieokreśloności doświadczenia (próby), którego wynik nie jest jednoznaczny (np. rzutu kości)". Cytat z: *Słownik wyrazów obcych i zwrotów obcojęzycznych*, W. Kopaliński [dostępny w Internecie: http://www.slownik-online.pl/kopalinski].

losowość symboli w alfabecie używanym przez źródło, tym więcej bitów informacji może ono wyprodukować. Wyższa wartość *H* odpowiada zatem większemu deficytowi danych.

Przejdźmy teraz do przedstawienia dwóch fundamentalnych dla teorii informacji twierdzeń Shannona. Rozważmy zakodowaną wiadomość, którą chcemy przesłać kanałem o określonej pojemności (przepustowości) *C*, określaną w bitach na sekundę. Twierdzenie o kanale bezszumowym stwierdza, że:

> Niech źródło ma entropię *H* (bitów na symbol) oraz niech kanał komunikacyjny ma przepustowość *C* (bitów na sekundę). Wtedy możliwe jest zakodowanie informacji wysyłanej ze źródła przez ten kanał w taki sposób, że średnia prędkość transmisji wynosi *C/H* – ε symboli na sekundę, gdzie ε jest dowolnie małe. Niemożliwa jest transmisja z prędkością większą niż *C/H* (Shannon, Weaver, 1998, s. 59).

Zatem można transmitować symbole przez bezszumowy kanał z prędkością dowolnie bliską *C/H*, jednak niezależnie od tego, jak dobre jest kodowanie, średnia ta nigdy nie przekroczy *C/H*[14].

Spróbujmy teraz przeanalizować związki pomiędzy szumem a ilością informacji w omawianym procesie komunikacji. Gdy podczas przesyłania komunikatu pojawią się zakłócenia (szum), komunikat może ulec zniekształceniu, zafałszowaniu, może także zawierać błędy oraz pewne „nadmiarowe" (pochodzące z zewnątrz) elementy. Jeśli jednak miara informacji zależy od stopnia nieprzewidywalności komunikatu, to im więcej szumu w kanale komunikacyjnym, tym więcej informacji niesie komunikat. Skoro wzrasta niepewność – rośnie też ilość informacji. Czy zatem szum jest zjawiskiem pożądanym? Weaver (1949) pisze:

> Prawdą jest, że gdy występuje szum, to odbierany sygnał jest wybrany z bardziej zróżnicowanego zbioru symboli niż ten zamierzony przez nadawcę. Ta sytuacja dobrze oddaje pułapkę semantyczną, w jaką można popaść, jeśli nie pamięta się, że 'informacja' jest używana tutaj w specjalnym znaczeniu, mierzy wolność wyboru, a tym samym niepewność co do tego, jaki wybór został dokonany (s. 32).

Oczywiście, wbrew temu, co można by wnioskować z definicji informacji, niepewność związana z zakłóceniami transmisji (z szumem) jest zjawi-

---

[14] Niezwykłość odkrycia Shannona polega na tym, że przedstawione w twierdzeniu dolne ograniczenie daje się osiągnąć. Niestety, sam dowód tego faktu jest niekonstruktywny i pokazuje jedynie, że taki kod istnieje, a nie to, jak wygląda.

skiem negatywnym. Transmisja jest trudniejsza w przypadku występowania szumu. Jednak twierdzenie o kanałach z szumem głosi:

> Dany jest dyskretny kanał o przepustowości *C* oraz dyskretne źródło o entropii *H*. Jeśli $H \leq C$, to istnieje system kodowania taki, że sygnał źródła może być transmitowany przez kanał z dowolnie małą częstością błędów (lub z dowolnie małą dwuznacznością). Jeśli $H > C$, to możliwe jest zakodowanie sygnału ze źródła w taki sposób, że dwuznaczność jest mniejsza niż $H-C+\varepsilon$, gdzie ε jest dowolnie małe. Nie istnieje metoda kodowania, która daje mniejszą dwuznaczność niż $H-C$ (Shannon, Weaver, 1998, s. 71).

Powyższe twierdzenie mówi, że jeśli kanał komunikacyjny może przetransmitować tyle samo lub więcej informacji, niż produkuje źródło, to możliwe jest opracowanie efektywnej metody kodowania i transmitowania wiadomości z dowolnie małym prawdopodobieństwem błędu.

Na czym polega trudność w procesie komunikacji z szumem? Nie jest ona związana, jak pokazuje powyższe twierdzenie, z samym przesyłaniem informacji, lecz z efektywnością procesu komunikacji.

Weaver (1949) zauważa, że: „W komunikacji wydają się występować problemy na trzech poziomach: 1) technicznym, 2) semantycznym oraz 3) na poziomie wpływu" (s. 28). Problemy semantyczne związane są z interpretacją znaczenia komunikatu dokonaną przez jego odbiorcę – w porównaniu z zamierzonym znaczeniem nadanym mu przez nadawcę. Zagadnienia wpływu (lub efektywności) natomiast dotyczą szeroko pojętego oddziaływania informacji na działania zmierzające w pożądanym kierunku, na przykład – na ludzkie zachowania.

Matematyczna teoria komunikacji opisuje jej poziom techniczny. Aby móc mówić o jej aspekcie pragmatycznym (na poziomie 3), Weaver proponuje dodanie do przedstawionego wcześniej modelu komunikacji (por. rys. 3) trzech dodatkowych elementów: odbiorcy semantycznego (pomiędzy odbiornikiem a punktem docelowym), szumu semantycznego oraz audytorium (ang. *audience*). Odbiorca semantyczny po raz drugi odkodowuje komunikat, dokonując jego „semantycznej interpretacji", po to, by móc w pełni określić jego zawartość informacyjną i wykorzystać ją w swoim działaniu. Powinien on móc rozpoznać szum, nieunikniony w kanale komunikacyjnym ze względów technicznych, by oddzielić go od znaczącej części wiadomości. Szum semantyczny natomiast ma reprezentować zniekształcenia znaczenia wprowadzone przez źródło informacji, na przykład mówiącego, które – choć niezamierzone – wpływają na miejsce przeznaczenia, czyli na słuchającego. Szum ten powinien być rozpoznawany przez odbiornik oraz interpretowany przez odbiorcę jako element procesu komunikacji. Należy tak zorganizować lub przystosować wiadomość oryginalną, by suma

jej znaczenia i szumu semantycznego była równa znaczeniu wiadomości oczekiwanemu w miejscu przeznaczenia. Dzięki temu odbiorca, wyposażony w odpowiednie techniki dekodowania, mógłby odczytać jej właściwe znaczenie, a to poprawiłoby efektywność oddziaływania tej wiadomości.

W przedstawionym powyżej modelu komunikacji ostatnim elementem jest punkt docelowy, którym może być zarówno człowiek, jak i urządzenie. Weaver stwierdza, że w każdym procesie komunikacyjnym należy uwzględnić możliwość istnienia wielu odbiorców – czyli audytorium, które ma swoją pojemność (ang. *capacity*). Przepełnienie, zarówno kanału informacyjnego, jak i audytorium – na przykład poprzez nadmiarowość sygnałów lub skomplikowany sposób kodowania – prowadzić może do błędów w interpretacji, a tym samym do spadku efektywności komunikacji.

Matematyczna teoria komunikacji nie uwzględnia szczegółowo ani zaproponowanych przez Weavera dodatkowych elementów modelu komunikacyjnego, ani też kontekstu koniecznego do właściwej interpretacji znaczenia wiadomości[15]. Czym zatem jest opisywana przez nią informacja?

Najprostszą odpowiedzią na to pytanie wydaje się stwierdzenie, że jest to byt materialny przesyłany od nadajnika do odbiornika. Można jednak także twierdzić inaczej: informacja – w ujęciu Shannona i Weavera – to nie obiekt, lecz relacja (Heyck, 2008a, 2008b); przesyłać informację natomiast znaczy tworzyć związek pomiędzy nadajnikiem i odbiornikiem, a nie: przesyłać jakiś obiekt z jednego miejsca na drugie. To, co podróżuje przez kanał komunikacyjny, to wzorzec (relacja), który jest reprodukowany na miejscu przeznaczenia. A zatem to wzorce niosą informację. Weaver (1949) z kolei tak charakteryzuje informację opisywaną w stworzonej przez niego teorii:

> Pojęcie informacji rozwinięte w tej teorii wydaje się na pierwszy rzut oka rozczarowujące oraz dziwaczne – rozczarowujące, ponieważ nie ma nic wspólnego ze znaczeniem, natomiast dziwaczne, ponieważ nie dotyczy ono pojedynczej wiadomości, lecz statystycznego charakteru całego zestawu wiadomości; jest dziwaczne również dlatego, że w ramach warunków statystycznych obydwa słowa 'informacja' oraz 'niepewność' okazują się swoimi parametrami (s. 36).

Podsumowując, warto podkreślić, że teoria stworzona przez Shannona i Weavera nie jest teorią informacji w zwykłym sensie, ponieważ bada jedynie jej aspekt ilościowy, ograniczając się do analiz na poziomie syntaktycznym – zajmuje się bowiem komunikatami złożonymi z niezinterpretowanych symboli. Całkowicie ignoruje kontekst tych komunikatów. Odpowiedź „tak" zawiera tę samą ilość informacji, zarówno jako odpowiedź na pytanie:

---

[15] Weaver zdawał sobie sprawę ze znaczenia kontekstu, w którym wiadomość jest przesyłana, jako warunku określającego znaczenie komunikatu (por. Weaver, 1949).

„Czy wypijesz herbatę?", jak i jako odpowiedź na pytanie: „Czy wyjdziesz za mnie?". Co więcej, największą ilość informacji produkuje tekst, w którym prawdopodobieństwo wystąpienia każdego symbolu jest takie samo, to znaczy zupełnie losowy ciąg, zazwyczaj nic nie znaczących liter. Jest to zatem teoria informacji bez „znaczenia". Informacja minus znaczenie to „dane", wydaje się zatem, że bardziej odpowiednią nazwą dla tej teorii – niż powszechnie używana „teoria informacji" – wydaje się być „matematyczna teoria przesyłania danych".

Nie jest to jednak jedyne matematyczne ujęcie informacji.

3.2.2. Algorytmiczna teoria informacji

Przedstawiona powyżej matematyczna teoria komunikacji Shannona nie pozwala na określenie ilości informacji zawartej w danym obiekcie w oderwaniu od jego źródła, a w szczególności od prawdopodobieństwa wybrania danego symbolu z określonego zbioru. Możliwość określenia ilości informacji w pojedynczym obiekcie, poprzez badanie jego struktury, daje teoria, którą stworzyli niezależnie od siebie: Solomonoff, Kołmogorow i Chaitin w latach 60. XX wieku. Obecnie znana jest ona pod nazwą „algorytmiczna teoria informacji" lub „teoria złożoności algorytmicznej". Jej początków należy szukać w pracy Solomonoffa *A Preliminary Report on a General Theory of Inductive Inference*, zaprezentowanej po raz pierwszy na konferencji w Caltech w roku 1960 i wydanej drukiem w tym samym roku.

Solomonoff (1960) zdefiniował prawdopodobieństwo *a priori* skończonego ciągu symboli w następujący sposób: „Prawdopodobieństwo *a priori* ciągu otrzymuje się, rozważając uniwersalną maszynę Turing[16], której wyjściem jest rozważany ciąg. Przybliżenie tego prawdopodobieństwa dane jest przez najkrótsze wejście dla tej maszyny, które daje pożądany wynik" (Solomonoff, 1960, s. i). Definicję tę autor wykorzystuje do określenia złożoności obiektu: „Rozważmy bardzo długi ciąg symboli – na przykład fragment tekstu w języku angielskim lub długi wywód matematyczny. Będziemy rozważali taki ciąg symboli jako 'prosty' i posiadający duże prawdopodobieństwo *a priori*, jeśli istnieje bardzo krótki opis tego ciągu" (Solomonoff, 1960, s. 1). Długość owego opisu można interpretować, podobnie jak robili to Kołmogorow i Chaitin, jako długość programu generującego dany ciąg.

Kołmogorow (1965) zauważa, że korzystając z ujęcia Shannona, nie sposób obliczyć ilości informacji zawartej na przykład w książkach lub mapach. Stwierdza on, że „[w] rzeczywistości o wiele bardziej owocne jest

[16] Pojęcie maszyny Turinga zostało omówione w podrozdziale 1.2.

rozważanie ilości informacji ‘dostarczanej przez obiekt’ (x) ‘na temat obiektu’ (y)” (s. 4). Co prawda, rzeczywiste obiekty, które badamy, są bardzo złożone, ale można dokonać pewnych uproszczeń. Na przykład, rozważając wspomnianą już mapę (x) jako niosącą informacje na temat jakiegoś regionu (y), pominąć można nieistotne detale jej dotyczące – na przykład strukturę papieru czy rodzaj tuszu, którym została ona wydrukowana – ponieważ nie mają one wpływu na te informacje.

Kołmogorow zdefiniował względną złożoność obiektu y z x jako minimalną długość $l(p)$ programu p dla otrzymania y z x. Złożoność ta zależy od metody programowania, rozumianej jako funkcja $\varphi(p, x) = y$ łącząca obiekt y z programem p i obiektem x. Przy czym konieczne jest przyjęcie założenia, że funkcja φ jest częściowo rekurencyjna, to znaczy, że dla każdej takiej funkcji mamy:

$$K_{\varphi}(y/x) = \begin{cases} \min\limits_{\varphi(p,x)=y} l(p) \\ \infty, \text{jeśli nie istnieje takie } p, \text{że } \varphi(p,x) = y \end{cases}$$

Warto w tym miejscu zauważyć, że funkcje częściowo rekurencyjne nie są zdefiniowane „wszędzie” oraz że nie istnieje ustalona metoda stwierdzania, czy zastosowanie programu p do obiektu k prowadzi do wyniku, czy też nie. Dlatego też funkcja $K_{\varphi}(y/x)$ nie może być efektywnie obliczona (ogólnie rekurencyjna), nawet jeśli wiadomo, że jest skończona dla wszystkich x oraz y. Takie pojęcie informacji ma jedną zasadniczą wadę, o której pisał już sam jej autor: nie uwzględnia ona „trudności” stworzenia programu p do przechodzenia z obiektu x do y[17].

Przejdźmy teraz do algorytmicznej teorii informacji w ujęciu Chaitina (1966, 1969, 1977). Rozważa on maszyny Turinga z trzema taśmami: taśmą programu, roboczą oraz wynikową. Z taśmy programu, przesuwającej się tylko w jedną stronę, głowica maszyny odczytuje symbole 0 lub 1 (nie ma natomiast możliwości pisania na niej). Z taśmy roboczej natomiast (początkowo pustej), przesuwającej się w lewo lub w prawo, głowica czyta lub usuwa symbole (0, 1 i symbol pusty). Na taśmie wynikowej (również początkowo pustej), przesuwającej się tylko w jedną stronę, mogą być zapisywane symbole 0, 1 oraz przecinek, których nie można nadpisać, to znaczy zastąpić innymi. Każda maszyna Turinga tego typu ma skończoną liczbę n stanów, a jej działanie można zdefiniować za pomocą tabeli o wymiarach $n \times 3$[18].

---

[17] W pracy z roku 1965 Kołmogorow zapowiadał dalsze prace zmierzające do poszerzenia pojęcia złożoności tak, by ową „trudność” uwzględnić.

[18] Dokładny opis takich maszyn, w tym tabele opisujące ich programy, przedstawia Chaitin (1966).

Dla takich właśnie maszyn Chaitin definiuje pojęcia: prawdopodobieństwa *P*, entropii *H* oraz złożoności *I*.

P(*s*) to prawdopodobieństwo tego, że jeśli komórki na taśmie programu maszyny *M* wypełnione są losowo zerami i jedynkami (przez niezależne, losowe rzuty monetą), to maszyna *M* zatrzyma się, a na taśmie wynikowej znajdować się będzie skończony ciąg symboli (tak zwany string) *s*. Tak zdefiniowane prawdopodobieństwo pojawia się we wzorze na entropię:

$$H = -P(s)\ log_2\ P(s).$$

Przez program *p* autor rozumie taki string, który znajduje się na taśmie programu maszyny *M*, rozpoczynającej obliczenia od przeczytania jego pierwszego symbolu, która po przeczytaniu całego stringu *p* (z pominięciem ewentualnych innych symboli na taśmie) zatrzymuje się (kończy działanie). Program *p* natomiast Chaitin nazywa minimalnym, jeśli dla maszyny *M* nie istnieje krótszy program dający taki sam wynik jak *p*.

I z kolei, złożoność *I*(*s*) to najmniejsze *n* takie, że dla pewnej zawartości taśmy programu maszyna *M* zatrzymuje się, dając wynik *s,* po przejściu dokładnie *n* komórek na tej taśmie. Zatem *I*(*s*) jest rozmiarem minimalnego programu dla *s*.

A zatem *P* jest prawdopodobieństwem, że maszyna *M* oblicza *s* za pomocą losowego programu, natomiast *I* jest najmniejszą liczbą bitów konieczną do opisu algorytmu obliczającego *s* dla *M*.

Chaitin opisuje uniwersalną maszynę Turinga *U,* symulującą taką konkretną maszynę *M*, której identyfikator (numer) został dodany jako prefiks programu dla *U*. Maszyna *U* działa w następujący sposób: gdy na jej taśmie programu znajduje *i* zer zakończonych jedynką, wówczas symuluje obliczenia *i*-tej maszyny Turinga, czytając znajdujące się na tej taśmie symbole, następujące po jedynce[19]. Dla takiej właśnie maszyny zdefiniowane wcześniej wielkości *P, I* oraz *H* zyskują nazwy: *P*(*s*) – to prawdopodobieństwo algorytmiczne ciągu *s*, *H*(*s*) – entropia algorytmiczna, oraz *I*(*s*) – informacja algorytmiczna. Mogą być one nazwane, zgodnie z koncepcją Salomonoffa (1960), wielkościami *a priori*.

Powyższą charakterystykę *P, H* oraz *I* dla pojedynczego stringu *s* można uogólnić dla przypadku skończonego ciągu stringów, które na taśmie wynikowej oddzielane są przecinkami. Definiuje się wtedy łączne (ang. *joint*) prawdopodobieństwo $P(s_1, s_2, \dots, s_n)$, oraz łączną entropię $H(s_1, s_2, \dots, s_n)$

---

[19] Przez *i*-tą maszynę Turinga rozumie się tę, która znajduje się na liście wszystkich możliwych maszyn na *i*-tym miejscu, przy czym lista ta powstaje poprzez uszeregowanie definiujących te maszyny tablic według ich rozmiaru (to znaczy liczby stanów maszyny) oraz leksykograficznie, gdy rozmiar tablic jest taki sam.

i złożoność $I(s_1, s_2, \dots, s_n)$ określoną dla ciągów postaci $s_1, s_2, \dots, s_n$. Można wtedy określić również prawdopodobieństwo warunkowe $P(t_1, t_2, \dots, t_m|s_1, s_2, \dots, s_n)$ ciągu $t_1, t_2, \dots, t_m$ względem danego ciągu $s_1, s_2, \dots, s_n$, jako stosunek łącznego prawdopodobieństwa $P(s,t)$ do $P(s)$. Podobnie definiuje się entropię warunkową, otrzymując związek: $H(s,t) = H(s) + H(s|t)$. Chaitin nie poprzestaje na analizie dotyczącej ciągów skończonych; w 1977 roku formułuje on algorytmiczną teorię informacji dla obliczeń nieskończonych[20].

Chociaż każdy z autorów algorytmicznej teorii informacji – Solomonoff, Kołmogorow i Chaitin – miał różne motywacje do jej stworzenia, to jednak teorie ich są bardzo podobne. Solomonoffa interesowały kwestie filozoficzne związane z indukcją, a do ich formalnego opisu wykorzystał on prawdopodobieństwo *a priori* skończonego ciągu symboli. Z kolei Kołmogorow i Chaitin zdefiniowali złożoność ciągu jako długość najkrótszego programu generującego ten ciąg, zgodnie z założeniem, że „[…] im krótszy program, tym większe uporządkowanie można przypisać ciągowi" (Chaitin, 1965, s. 38), a tym samym mniejsza jest jego złożoność. Teorie te, w odróżnieniu od teorii komunikacji Shannona, pozwalają na obliczenie ilości informacji zawartej w pojedynczych obiektach – przy użyciu pojęcia ich złożoności. Na przykład rozwinięcie dziesiętne liczby π ma bardzo niską złożoność (chociaż jest nieskończone), ponieważ istnieje bardzo prosty program generujący dowolną liczbę cyfr tego rozwinięcia.

Algorytmiczna teoria informacji zdobyła szybko popularność – jako podstawowa teoria informacji. Niektórzy twierdzą nawet, że złożoność Kołmogorowa jest bardziej fundamentalna niż opisana wcześniej entropia Shannona. Tak dużej popularności nigdy nie zdobyła przedstawiona poniżej teoria stworzona przez Polaka – Mariana Mazura.

3.2.3. Teoria informacji Mazura

Marian Mazur stworzył oryginalną teorię informacji, którą opisał w wydanej w 1970 roku książce *Jakościowa teoria informacji*. Koncepcja ta, wywodząca się z cybernetyki, wiąże istotę informacji z procesami komunikacyjnymi i sterującymi zachodzącymi w złożonym układzie. Nie jest to zatem teoria informacji samej w sobie, lecz teoria informowania poprzez transmisję komunikatów. Jej autor zdefiniował sześć sposobów informowania wiernego, a także przeanalizował wszelkie możliwe rodzaje zniekształceń informacji.

---

[20] Wykorzystuje w tym celu wyniki Roberta M. Solovaya.

Mazur zauważył, że teoria Shannona nie tylko ogranicza się do jednego tylko aspektu informacji – do mierzenia jej ilości – ale również nie odpowiada na pytanie o ilość informacji w zwykłych, spotykanych na co dzień sytuacjach. Określenie „ilości informacji" wymaga bowiem zawsze określenia zbioru wszystkich możliwych komunikatów, co jest niezbędne do obliczenia ich prawdopodobieństwa. Jednak w praktyce często bywa to niemożliwe. Rozważmy przykłady podane przez Mazura.

Programy nauczania historii w szkole zawierają mniejszą lub większą liczbę faktów historycznych. Dla każdego nauczyciela jasne jest, że przy obszerniejszym programie uczeń otrzymuje więcej informacji niż przy programie uboższym. Jednak do ilościowego określenia tej różnicy nie można posłużyć się „ilością informacji" w sensie Shannona, ponieważ nie sposób określić liczby wszystkich faktów historycznych; a co więcej – nie ma sensu mówienie o prawdopodobieństwie zdarzeń historycznych, które już się wydarzyły. Mazur (1970) zauważa również, że:

> Z podobnych przyczyn niepodobna określić „ilość informacji" zawartą np. w stwierdzeniu, że teraz jest rok 1970. Do zbioru ilu lat należy bowiem wymieniony rok? Jak można mówić o prawdopodobieństwie występowania poszczególnych lat, zwłaszcza lat minionych, a więc niemogących się już zdarzyć z żadnym prawdopodobieństwem? (s. 17).

Pojęcia ilości informacji Shannona nie można również zastosować do mapy[21], chociaż niewątpliwie zawiera ona jej wiele, na przykład że Poznań leży na zachód od Warszawy oraz że odległość pomiędzy tymi miastami wynosi 303 km. Ile bitów zawiera każda z tych informacji? O jakie prawdopodobieństwo może w tym przypadku chodzić? Nie wiemy również, jak określić ilość informacji zawartą w twierdzeniach matematyki, na przykład – w informacji o związku pomiędzy długością średnicy koła a długością jego promienia.

Mazur próbuje uniknąć tej trudności, ograniczając się w swoich rozważaniach do zjawisk związanych z dążeniem do pewnego celu, czyli – do sterowania. Dzięki temu w tworzonej teorii będzie mógł użyć terminologii z zakresu cybernetyki, w tym pojęcia transformacji komunikatów, z rozróżnieniem kodów (jako transformacji wzdłuż toru sterowniczego) oraz informacji (jako transformacji w poprzek tego toru). Przeanalizujmy teraz podstawowe definicje.

Do analizy procesu sterowniczego konieczne jest wprowadzenie następujących pojęć cybernetycznych:

---

[21] Przykład mapy omawiał również Kołmogorow (por. paragraf 3.2.2).

– *źródło oddziaływania* – system oddziałujący na inny system w obwodzie sterowniczym,
– *odbiornik oddziaływania* – system, na który oddziałuje inny system w obwodzie sterowniczym,
– *tor sterowniczy* – system, za którego pośrednictwem pewien system oddziałuje na inny system (zaczyna się on na wyjściu źródła oddziaływania, a kończy na wejściu odbiornika oddziaływania).

Rozważmy dla przykładu działanie pieca wyposażonego w automatyczny regulator temperatury. Gdy temperatura pieca wzrasta ponad oznaczoną wartość, regulator przerywa dopływ energii elektrycznej – powodując jego wyłączenie – natomiast gdy temperatura stygnącego pieca spadnie poniżej określonej wartości, regulator wówczas dopływ ten przywraca. Z punktu widzenia użytkownika, regulator jest źródłem oddziaływania, a piec jego odbiornikiem.

Na oddziaływanie w torze sterowniczym składa się pewna liczba stanów fizycznych nazywanych komunikatami. Mazur (1970) definiuje komunikat jako „stan fizyczny różniący się w określony sposób od innego stanu fizycznego w torze sterowniczym" (s. 35). Wyróżnia przy tym dwa rodzaje zbiorów komunikatów:
– zbiór poprzeczny – zbiór komunikatów w dowolnym miejscu toru sterowniczego,
– zbiór wzdłużny – zbiór komunikatów, które powstały z innych komunikatów lub z których powstały inne komunikaty, przy czym każdy z komunikatów tego zbioru należy do innego zbioru poprzecznego.

Mamy zatem do czynienia ze zbiorem fizycznych stanów toru sterowniczego (tak zwanych stanów poprzecznych), składającym się z komunikatów. Pierwszy komunikat nazywa się przy tym oryginałem, a ostatni – obrazem. W obrębie takiej struktury można mówić o „informacji" i „informowaniu".

Dla zdefiniowania kluczowego pojęcia „informacja" potrzebne są jeszcze trzy inne określenia: „asocjacja komunikatów" (nieuporządkowana para komunikatów wyodrębnionych ze wzdłużnego lub poprzecznego zbioru komunikatów w procesie sterowniczym), „asocjacja informacyjna" (asocjacja komunikatów z poprzecznego zbioru) oraz „transformacja" (proces, jakiemu należy poddać jeden z komunikatów asocjacji, aby otrzymać drugi). Transformacja określa zatem związek między komunikatami asocjacji. Mazur zwraca uwagę, że istnieje wiele transformacji, które mogą przekształcać dany komunikat w inny. Na przykład, komunikat „a = 2" można przekształcić w komunikat „b = 8" za pomocą transformacji: $a + 6 = b$, $4a = b$ oraz $3a + 2 = b$.

Po przedstawieniu koniecznych pojęć autor podaje następującą definicję (Mazur, 1970, s. 70): „Informacja jest to transformacja jednego komunikatu

asocjacji informacyjnej w drugi komunikat tej asocjacji". Natomiast dalej dodaje:

> Informacja jako transformacja jednego komunikatu w drugi (na przykład oryginału w inny oryginał, bądź obrazu w inny obraz[22]) jest związkiem pomiędzy dwoma komunikatami i w takim sensie można mówić, że informacja jest zawarta w komunikatach (oryginałach, obrazach). Biorąc to pod uwagę, można wprowadzić ponadto konwencję terminologiczną: informowanie jest to transformacja informacji zawartej w oryginałach w informację zawartą w obrazach. (s. 121)

W dążeniu do określenia „liczby informacji" Mazur wyróżnia trzy jej rodzaje:

– użyteczną – informacja spośród najmniejszej możliwej ich liczby zawartych w łańcuchu informacyjnym[23], niezbędnych w danych procesie sterowniczym,
– redundancyjną – wynikającą z innych informacji danego łańcucha informacyjnego, oraz
– pasożytniczą – pochodząca spoza danego procesu sterowniczego.

Pomija przy tym dwa ostatnie rodzaje, koncentrując się na informacji użytecznej, którą podzielił na opisującą i identyfikującą. Wykazał ponadto, że tylko „liczba informacji identyfikujących" jest tym samym, co „ilość informacji" wyrażona wzorem Shannona – wbrew panującemu dotychczas przeświadczeniu, że matematyczna teoria komunikacji odnosi się do wszelkich informacji. Prześledźmy rozważania, które doprowadziły go do takiego wniosku.

Informację opisującą Mazur definiuje jako informację spośród najmniejszej możliwej liczby informacji, niezbędnych do określenia *dowolnego* komunikatu w łańcuchu informacyjnym. I dalej, informacja identyfikująca to informacja spośród najmniejszej możliwej liczby informacji niezbędnych do określenia komunikatu *wyróżnionego* z łańcucha informacyjnego. Do wyróżnienia jednego komunikatu z takiego łańcucha konieczne jest pewne, możliwe do sprawdzenia, kryterium. Mogą nim być: czas (określenie chwili, w jakiej występuje wyróżniony komunikat), przestrzeń (miejsce, w którym on występuje), kolejność (numer komunikatu) i tym podobne.

---

[22] Oryginał jest to komunikat należący do zbioru poprzecznego komunikatów na wyjściu źródła oddziaływania, a obraz to komunikat należący do zbioru poprzecznego komunikatów na wejściu odbiornika oddziaływania. Na przykład: każdy pomiar jest procesem, w którym wartości mierzone są oryginałami, a wyniki pomiaru – obrazami, mapa z kolei (zbiór obrazów) jest odwzorowaniem terenu (zbiór oryginałów) – [przypis mój I. B-K].

[23] Łańcuch informacyjny jest to łańcuch komunikatów poprzecznego zbioru komunikatów.

Oznaczmy teraz przez *D* liczbę informacji opisujących potrzebnych do określenia jednego komunikatu. Mazur dowodzi następujących twierdzeń.

Twierdzenie 1. Liczba informacji opisujących jeden komunikat w łańcuchu informacyjnym składającym się z *n* różnych komunikatów jest równa liczbie *n* tych komunikatów.

Zastanówmy się teraz, jaka jest liczba informacji opisujących, potrzebnych do zidentyfikowania wyróżnionego komunikatu. Powróćmy w tym celu do opisanej wcześniej zabawy w odgadywanie komunikatów poprzez stawianie pytań. Przypuśćmy, że chcemy się dowiedzieć, jaka jest pora roku. Wiemy, że są cztery możliwe odpowiedzi: wiosna, lato, jesień, zima. A zatem liczba informacji opisujących, umożliwiających identyfikację wyróżnionego komunikatu (pory roku), zależy od kolejności ich uzyskiwania, to znaczy od kolejności stawiania pytań i otrzymywania odpowiedzi. W przypadku czterech komunikatów (czteroelementowego zbioru pór roku), w najgorszym przypadku potrzebne są trzy informacje (trzy pytania) – i to bez względu na brzmienie odpowiedzi na ostatnie z nich. Sytuacja w najgorszym przypadku jest bowiem następująca: pytanie pierwsze – „Czy jest wiosna?" , odpowiedź – „nie", pytanie drugie – „Czy jest lato?", odpowiedź – „nie", pytanie trzecie – „ Czy jest zima?". Po odpowiedzi na ostatnie pytanie wyróżniony komunikat jest już zidentyfikowany, jeśli bowiem brzmi ona „tak", to porą roku jest zima, a w przeciwnym przypadku jest nią jesień. Zatem do określenia jednego z czterech komunikatów potrzeba trzech informacji opisujących. Jest to pozornie niezgodne z twierdzeniem 1., mówiącym, że liczba informacji opisujących jeden komunikat w łańcuchu informacyjnym składającym się z 4 różnych komunikatów jest równa liczbie tych komunikatów – czyli 4. Należy jednak pamiętać, że informacja o tym, ile jest komunikatów (ile jest pór roku), jest z góry daną czwartą informacją. Co więcej:

Twierdzenie 2. Liczba informacji opisujących, potrzebnych do zidentyfikowania wyróżnionego komunikatu, jest jednoznacznie określona tylko w łańcuchu informacyjnym składającym się z dwóch komunikatów.

Tylko w przypadku pary komunikatów: „wypadł orzeł" oraz: „wypadła reszka", do zidentyfikowania wyróżnionego komunikatu wystarcza zarówno informacja, że „wypadł orzeł", jak i informacja: „nie wypadła reszka". Wobec tego nieistotne jest, czy rozpocznie się identyfikację od pytania: „Czy wypadł orzeł?", czy też od pytania: „Czy wypadła reszka?".

Można uogólnić ten wynik. Jednym ze sposobów identyfikacji elementów zbioru, opisanym podczas zabawy w zadawanie pytań, jest wielokrotne dzielenie tego zbioru na dwie części (o możliwie takiej samej liczbie elementów), aż do uzyskania pary komunikatów, a w niej komunikatu wyróżnionego. Poniższe twierdzenie określa, ile takich podziałów trzeba dokonać.

Twierdzenie 3. Liczba informacji identyfikujących jeden komunikat w łańcuchu informacyjnym składającym się z *n* różnych komunikatów może być określona przez logarytm dwójkowy liczby komunikatów, co można zapisać symbolicznie wzorem:

$$H_n = \log_2 n .$$

Twierdzenie 4. Jeżeli w łańcuchu informacyjnym, składającym się z *n* komunikatów, występuje *m* klas utworzonych odpowiednio z $n_a, n_b, ..., n_m$ jednakowych komunikatów, to średnią liczbę informacji identyfikujących można wyrazić wzorem

$$H = \frac{n_a}{n} \log_2 \frac{n}{n_a} + \frac{n_b}{n} \log_2 \frac{n}{n_b} + ... + \frac{n_m}{n} \log_2 \frac{n}{n_m} .$$

Twierdzenie 5. Średnia liczba informacji identyfikujących jest równa logarytmowi dwójkowemu średniej liczby informacji opisujących.

$$H = \log_2 D .$$

Z powyższego wzoru natychmiast wynika, że liczba informacji identyfikujących *H* jest mniejsza od liczby informacji opisujących *D*.

Korzystając z powyższych twierdzeń, porównajmy teraz pojęcia „liczby informacji" oraz „ilości informacji" Shannona. Przytoczmy w tym celu rozumowanie Mazura (1970, s. 202–203).

Prawdopodobieństwo wyraża się stosunkiem liczby zdarzeń wyróżnionych pod określonym względem (sprzyjających zajściu danego zdarzenia) do liczby wszystkich możliwych zdarzeń. Zatem jeżeli na *n* zdarzeń zdarzenie *a* może wystąpić n_a razy, zdarzenie *b* – n_b razy, …, zdarzenie *m* – n_m razy, to ich prawdopodobieństwa można wyrazić następująco:

$$P_a = \frac{n_a}{n} , \; P_b = \frac{n_b}{n} , ..., \; P_m = \frac{n_m}{n} .$$

Wtedy wzór wyrażający średnią liczbę informacji identyfikujących (na mocy twierdzenia 4.) przybiera postać:

$$H = -P_a \log_2 P_a - P_b \log_2 P_b - ... - P_m \log_2 P_m .$$

Nietrudno zauważyć, że jest on identyczny ze wzorem Shannona na entropię źródła:

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i .$$

Porównując te wzory, należy mieć na uwadze jednostki, w jakich wyrażane są opisywane przez nie wielkości. W teorii Shannona jeden bit to ilość informacji powstająca podczas wyboru jednego z dwóch jednakowo prawdopodobnych zdarzeń. W teorii Mazura pojęciu temu odpowiada „jedna informacja identyfikująca", wyróżniająca jeden z dwóch komunikatów – jednak bez zastrzeżenia, że muszą być one jednakowo prawdopodobne, ponieważ każdy z nich uważa się za jednorazowy. Jeśli jakiś komunikat powtarza się, to mówimy o klasie komunikatów; na przykład w rzucie monetą rozróżnienie wyników „wypadł orzeł" i „wypadła reszka" jest tylko sposobem mówienia o tym, że pewne wyniki zalicza się do klasy „orzeł", a inne do klasy „reszka".

Skoro „ilość informacji" wyrażona w bitach i „liczba informacji identyfikujących" oznaczają tę samą wielkość, to powstaje pytanie, dlaczego w pewnych sytuacjach – takich jak wspomniane już mapy bądź programy nauczania – pojęcie ilości informacji Shannona okazuje się nieprzydatne. Spróbujmy na przykład określić, jaką ilość informacji zawiera zależność pomiędzy długością średnicy koła a długością jego promienia. Aby to ustalić, posługując się przy tym pojęciem ilości informacji Shannona, trzeba by określić prawdopodobieństwo występowania promienia i średnicy koła, a następnie zastosować wzór na *H*. Jednak zarówno promień, jak i średnica koła są pojęciami abstrakcyjnymi, a nie zdarzeniami, które mogą występować z mniejszym lub większym prawdopodobieństwem. W teorii Mazura natomiast możliwe jest przedstawienie tej zależności w postaci dwóch informacji opisujących: 1) dany jest promień koła, 2) średnica jest dwa razy większa od promienia. Wtedy, znając liczbę informacji opisujących $D_2 = 2$, można obliczyć liczbę informacji identyfikujących $H_2 = \log_2 2 = 1$, co jest równoważne jednemu bitowi informacji.

Analizując powyższy przykład, można odnieść mylne wrażenie, że teoria Mazura rozwiązuje trudności matematycznej teorii komunikacji, wynikające z oparcia tej drugiej na prawdopodobieństwie. Sam jej autor zauważa, że co prawda w każdym procesie sterowniczym występują komunikaty oraz określające je informacje opisujące, jednak nie wszystkie sytuacje można sprowadzić do identyfikacji komunikatów, a tylko w takim przypadku może być mowa o liczbie informacji identyfikujących, czyli o ilości informacji. A zatem nie zawsze można zamienić „ilość informacji" na „liczbę informacji". I tak na przykład w twierdzeniach geometrii nie chodzi o liczbę informacji identyfikujących (nie stawia się tam pytań: „Czy spośród dwóch elementów, jakimi są promień i średnica, wystąpi promień czy średnica?"), ale o liczbę informacji opisujących, co znajduje wyraz w takich sformułowaniach jak: „dane są trzy boki trójkąta…", „jeśli promień kuli wpisanej w stożek…" i tym

podobne. Mazur (1970) stwierdza, że „[…] na podstawie liczby informacji opisujących można zawsze określić liczbę informacji identyfikujących (ilość informacji), ale ma to sens tylko tam, gdzie istotnie chodzi o identyfikację" (s. 206).

Podsumowując, warto zauważyć, że „informacja" rozważana w teorii Mazura traktowana jest jako wielkość fizyczna, ponieważ transformowanie komunikatów to proces fizyczny, oparty na przypływie energii. Co więcej, zaproponowana przez autora nazwa „jakościowa teoria informacji" może być bardzo myląca; występujący w niej termin „jakościowa" nie odnosi się bowiem do informacji, lecz do teorii na jej temat. Używając takiej terminologii, Mazur chciał zwrócić uwagę na to, że jego teoria odpowiada na pytania o naturę informacji: czym ona w istocie jest, jakie są jej rodzaje oraz na czym polegają procesy informowania. Całkowicie pomija jednak pojęcie jakości informacji – rozumiane jako jej wartość (związaną na przykład ze znaczeniem) – na której koncentrują się teorie semantyczne.

3.2.4. Teoria informacji semantycznej

Pojawienie się w latach 50. XX wieku matematycznych teorii informacji, a w szczególności teorii komunikacji Shannona, zostało przez filozofów przyjęte początkowo z entuzjazmem, który z czasem jednak osłabł. Obecnie większość z nich uważa, że teorie te wprowadzają zbyt surowe ograniczenia w rozważaniach semantycznych i pragmatycznych aspektów informacji, co prowadzi do znacznego zubożenia jej pojmowania. Badacze ci mają natomiast odmienne poglądy na temat siły i znaczenia tych ograniczeń.

Z filozoficznego punktu widzenia celowe wydaje się analizowanie informacji nie tylko na poziomie syntaktycznym, jak to czynią teorie matematyczne, ale również – semantycznym. Postulat rozważania informacji wraz z jej „zawartością semantyczną" doprowadził do powstania semantycznych teorii informacji. Patrząc z perspektywy historycznej, pierwsze takie teorie były zależne od teorii Shannona, a późniejsze ewoluowały w stronę całkowitej autonomii. Jednak większość z nich, nawet teorie najbardziej niezależne od matematycznej teorii komunikacji, przejmuje z niej przynajmniej dwie rzeczy: model komunikacji oraz zasadę głoszącą, że ilość informacji niesionej przez dany komunikat jest odwrotnością jego prawdopodobieństwa[24].

Jedno z pierwszych określeń informacji semantycznej zaproponowali Bar-Hillel i Carnap (1952, 1953). Podana przez nich definicja sformułowana

[24] Barwise (1989) nazwał tę zasadę zasadą odwrotnej zależności (ang. *Inverse Relationship Principle*).

jest w terminach lingwistycznych[25] i wykorzystuje pojęcie prawdopodobieństwa indukcyjnego, wprowadzone przez Rudolfa Carnapa (Carnap, 1950, 1952, 1959). Przyjrzyjmy się sposobowi, w jaki autorzy definiują pojęcie treści (zawartości, ang. *content*) semantycznej.

Rozważmy ustalony język, w którym występuje skończona liczba stałych, oznaczających indywidua (rzeczy, wydarzenia, sytuacje), oraz skończona liczba predykatów jednoargumentowych, oznaczających ich pierwotne własności. Formuły atomowe (ang. *atomic statement*), na przykład P_1a_1 (a_1 ma własność P_1), stwierdzają, że własności pierwotne zachodzą dla indywiduów. Formuły atomowe oraz wyrażenia utworzone przez ich połączenie spójnikami zdaniowymi (negacją, alternatywą, koniunkcją, implikacją i równoważnością) są formułami molekularnymi (ang. *molecular statement*). Wszystkie formuły atomowe i ich negacje nazywane są formułami podstawowymi (ang. *basic statements*). Każda formuła jest albo L-prawdziwa (logicznie prawdziwa, analityczna), albo L-fałszywa (logicznie fałszywa, przeczy sama sobie), bądź też – rzeczowa (ang. *factual*, logicznie nieokreślona, syntetyczna).

Opisem stanu (ang. *state-description*) jest alternatywa, zawierająca jako składniki dla każdej formuły atomowej albo tę właśnie formułę, albo jej negację, oraz – niezawierająca innych formuł. A zatem opis stanu przedstawia możliwy stan rozważanego uniwersum. Dla każdej formuły *j* zbiór tych opisów stanów, w których *j* zachodzi (to znaczy takich, które L-implikują *j*), nazywa się zasięgiem (dziedziną, ang. *range*) *j*. Zasięg *j* jest pusty wtedy i tylko wtedy, gdy *j* jest L-fałszywa; w przeciwnym przypadku *j* jest L-równoważna alternatywie opisów stanów w jej zasięgu.

Niech teraz L_n^m będzie funkcyjnym systemem semantycznym pierwszego rzędu, który zawiera *n* indywiduów ($a_1, \dots, a_n$) oraz *m* własności pierwotnych ($P_1, \dots, P_m$). Element treści (ang. *content-element*) to alternatywa, w której dla każdej z $n \cdot m$ formuł atomowych występuje albo ta formuła, albo jej zaprzeczenie (nie równocześnie). Na przykład jednym z 64 elementów treści języka L_3^2 jest alternatywa postaci:

$$P_1a_1 \vee \neg P_2a_1 \vee \neg P_1a_2 \vee P_2a_2 \vee P_1a_3 \vee P_2a_3.$$

Zbiór wszystkich elementów treści L-implikowanych przez wyrażenie *i* (w L_n^m) nazywa się treścią tego wyrażenia i oznacza przez Cont(*i*). Bar-Hillel i Carnap (1953) tak wyjaśniają znaczenie tego pojęcia: „Proponujemy Count(*i*) jako eksplikację dla zwykłego pojęcia 'informacji dostarczanej przez formułę *i*' w jego sensie semantycznym" (s. 149).

---

[25] Podobną, ale nie identyczną teorię informacji, w której pojęcie informacji interpretowane jest w ramach lingwistycznych, przedstawił Fred Dretske (1981, 1983).

Miarą treści semantycznej zawartej w p jest dopełnienie prawdopodobieństwa *a priori* P(p):

$$\text{Cont}(p)=1\text{-P}(p).$$

Również „informatywność" (nazywana „miarą semantycznej nowości") ma w tej teorii swoje określenie:

$$\text{INF}(p) = \log \frac{1}{1-\text{Cont}(p)} = -\log \text{P}(p).$$

Takie probabilistyczne rozumienie treści informacji było rozwijane później, między innymi przez Hintikkę (1970) i Dretske'a[26] (1981, 1983).

Prawdopodobieństwo występujące w tych wzorach może mieć różne interpretacje. Bar-Hillel i Carnap w pracy z 1953 roku pokazali, że rozkład prawdopodobieństwa może być wynikiem konstrukcji logicznej formuł atomowych w wybranym języku formalnym. Powoduje to bezpośrednią zależność pomiędzy językiem obserwacyjnym a formalnym. Z kolei Dretske odnosi wartość prawdopodobieństwa do obserwowanego stanu rzeczy (s):

$$I(s) = -log\ P(s),$$

gdzie $I(s)$ oznacza informację zawartą w s.

Informację semantyczną można również rozumieć w terminach modalnych. Treścią semantyczną zdania jest wówczas zbiór wszystkich możliwych światów, w których zdanie to jest fałszywe – czyli światy wykluczone przez to zdanie. Oznaczając przez W zbiór wszystkich możliwych światów, Sequoiah-Grayson (2007) definiuje treść semantyczną w następujący sposób (s. 335):

$$\text{Cont}(s)=_{def} \{x \in W : x \models \neg s\}$$

Łatwo zauważyć, że powyższa teoria informacji semantycznej zachowuje wspomnianą wcześniej zasadę odwrotnej zależności[27], głoszącą, że ilość informacji niesionej przez dany komunikat jest odwrotnością jego prawdopodobieństwa. Związana jest z nią pewna własność informacji – nazywana obecnie paradoksem Bar-Hillela–Carnapa – głosząca, że zdania sprzeczne, których prawdopodobieństwo wynosi zero, niosą najwięcej informacji (również semantycznej). Oto, co piszą Bar-Hillel i Carnap w pracy z roku 1953:

---

[26] Szczegółowe omówienie teorii Dretske'a znaleźć można np. w *Epistemologii informacji* (Hetmański, 2013).

[27] Zasada ta, pochodząca z matematycznej teorii komunikacji, jest przyjmowana przez większość teorii informacji semantycznej (por. przypis 23).

> To może wydawać się na pierwszy rzut oka dziwne, że zdania wewnętrznie sprzeczne (ang. *self-contradictory*), a więc takie, których żaden idealny odbiorca nie akceptuje, uważa się za niosące najwięcej informacji. Należy jednak podkreślić, że informacja semantyczna nie jest tutaj rozumiana jako implikująca prawdziwość. Fałszywe zdanie, któremu zdarza się mówić wiele, jest w naszym rozumieniu bardzo informatywne. Nie interesuje nas, czy informacje, które niesie, są prawdziwe czy fałszywe, wartościowe naukowo czy też nie. Zdania sprzeczne zakładają zbyt wiele; są zbyt informatywne, by być prawdziwe (s. 229).

Powyższa własność była wielokrotnie dyskutowana w literaturze przedmiotu. Niektórzy badacze, podobnie jak Bar-Hillel i Carnap, traktowali ją jako pewną nieodłączną konsekwencję teorii słabej[28] informacji semantycznej i nie nazywali paradoksem. Inni szukali jego rozwiązania. Pojawiły się różne propozycje: od przyjęcia, że wszystkie przypadki sprzeczne niosą taką samą – albo nieskończoną, albo zerową – ilość informacji, poprzez wyeliminowanie *a priori* z rozważań wszystkich przypadków sprzecznych, aż po teorię informacji silnie semantycznej – jedną z obecnie najbardziej wpływowych, stworzoną przez Luciano Floridiego (Floridi, 2003, 2004a, 2004c, 2005a, 2005b, 2005c).

3.2.5. Teoria informacji silnie semantycznej Floridiego

Rozważmy za Floridim (2004c) sytuację, która ułatwi nam zrozumienie przedstawionych pojęć. Pewnego ranka wsiadasz do samochodu i przekręcasz kluczyk w stacyjce. Jednak zamiast spodziewanego dźwięku silnika słyszysz ciszę. Zauważasz ponadto, że pali się czerwona lampka ze znaczkiem akumulatora, co przypomina ci, że wczoraj zapomniałeś wyłączyć światła. Telefonujesz do mechanika, który mówi ci, że w instrukcji obsługi samochodu znajdziesz wskazówki, co należy zrobić, by uruchomić samochód z użyciem kabli i akumulatora w innym aucie. Pożyczasz od sąsiada potrzebne kable, analizujesz opis i ilustracje w instrukcji – i w końcu rozwiązujesz problem. W jaki sposób można opisać formalnie występujące w opisie tej sytuacji pojęcie informacji?

Floridi (2005b) podaje następującą ogólną definicję informacji:

(GDI) σ jest instancją informacji, rozumianej jako zawartość semantyczna, wtedy i tylko wtedy, gdy:

(GDI.1) σ składa się z jednej lub więcej *danych*;

---

[28] W odróżnieniu od silnej informacji semantycznej, której teorię (opisaną w paragrafie 3.2.5) stworzył Luciano Floridi.

(GDI.2) dane w σ są *poprawnie sformułowane*;

(GDI.3) poprawnie sformułowane dane w σ są *znaczące*[29].

Powyższa definicja wskazuje trzy podstawowe, zdaniem Floridiego, cechy informacji: (1) składa się z ona z określonej liczby danych, (2) dane te są poprawnie sformułowane (to znaczy utworzone zgodnie z regułami pewnej składni systemu, języka lub kodu, który analizujemy) oraz (3) dane te są znaczące (ang. *meaningful*), czyli muszą mieć pewne znaczenie w używanym systemie czy języku.

W tym miejscu warto zwrócić uwagę, że tak rozumiana informacja semantyczna nie musi być językowa. Obrazy są często bardziej komunikatywne niż ich słowne opisy, jak to jest widoczne na przykład w instrukcjach obsługi czy montażu. Pojawiające się w powyższej definicji pojęcie „składni" autor rozumie zatem bardzo szeroko, nie tylko lingwistycznie (jako reguły składniowe języka), ale i bardziej ogólnie – jako coś, co wyznacza formę, strukturę czy konstrukcję obiektu; w tym sensie o składni mogą mówić inżynierowie, reżyserzy, malarze czy szachiści. W opisanej na wstępie sytuacji z samochodem istnieją reguły mówiące o tym, że ilustracja przedstawiająca sposób podłączenia kablami dwóch aut obrazuje (na płaszczyźnie) samochody stojące obok siebie, a nie na przykład jeden na drugim. Taka „obrazkowa składnia" powoduje, że ilustracja jest dla oglądającego (przynajmniej potencjalnie) zrozumiała. Jako składnię traktować można również właściwe podłączenie akumulatora (powodujące jego funkcjonowanie) oraz – poprawne używanie języka polskiego (umożliwiające porozumienie z sąsiadem w sprawie pożyczenia kabli).

Informacja w ujęciu Floridiego może składać się z czterech rodzajów danych. Pierwszy z nich to *dane podstawowe* (ang. *primary data*) – przechowywane na przykład w bazach danych – takie jak tablica liczb. W opisywanej sytuacji czerwona lampka świecąca się w samochodzie jest przykładem danej podstawowej wyrażającej informację na temat stanu akumulatora. W starszych publikacjach Floridiego pojawiają się również *dane poboczne* (*ponadpodstawowe*; ang. *secondary data*), które są w pewnym sensie przeciwnością danych podstawowych, gdyż powstają przez ich nieobecność (można by je nazwać anty-danymi). Może to zobrazować brak pojawiającego się zazwyczaj dźwięku po przekręceniu kluczyka w stacyjce, który informuje nas o tym, że silnik nie pracuje. A zatem cisza również może informować! Jest to jedna z cech informacji – jej brak może być informatywny. Kolejny rodzaj danych to *metadane*, czyli wskazówki dotyczące natury innych – zazwyczaj podstawowych – rodzajów danych. Opisują one takie

[29] Floridi w swoich publikacja używa również słowa „infon" dla określenia instancji informacji σ.

własności jak: lokalizacja, format, aktualizacja, dostępność, ograniczenia kopiowania i tym podobne. Metainformacja jest wówczas rozumiana jako informacja o naturze informacji, a więc na przykład: „'Akumulator jest wyczerpany' to informacja zapisana w języku polskim". Dane trzeciego rodzaju to *dane operacyjne* – związane z używaniem samych danych, z operacjami na całym systemie danych i z jego interpretacją. Powiązana z nimi informacja operacyjna dotyczy dynamiki systemu informacyjnego. Przypuśćmy, że samochód ma na tablicy rozdzielczej żółtą lampkę, która zapala się w sytuacji, gdy nie działa system sprawdzania funkcji samochodu. Jej światło może świadczyć o tym, że wskaźnik niskiego poziomu akumulatora nie funkcjonuje poprawnie, a tym samym nie wiadomo, czy prawdziwa jest hipoteza, że poziom energii w akumulatorze jest niski. Ostatni typ danych to *dane pochodne*, które można uzyskać z danych pozostałych rodzajów. Na przykład na podstawie analizy wydruku operacji na karcie kredytowej można uzyskać informację o tym, gdzie przebywał w danym dniu jej użytkownik (choćby dzięki temu, że płacił rachunek na stacji benzynowej).

Przyjrzyjmy się teraz podstawowym cechom informacji w ujęciu Floridiego. Przedstawiona przez niego definicja (GDI) wyraźnie wskazuje, że do istnienia informacji konieczne są dane, jednakże autor nie precyzuje ich typu. Można zatem przyjąć, że w najprostszym przypadku składa się ona z jednej danej (d), zredukowanej do braku identyczności pomiędzy dwoma znakami:

(Dd) $d = (x \neq y)$, gdzie x i y są niezinterpretowanymi zmiennymi.

Zatem informacja semantyczna nie zależy od typu danych. Jest ona również niezależna od swego nośnika, to znaczy obecny jej format czy język, w którym została zakodowana, nie są istotne. Ta sama informacja semantyczna może być cyfrowa lub analogowa, zapisana na papierze bądź wyświetlona na ekranie komputera, zapisana po polsku, angielsku lub w innym języku, wyrażona słowami lub rysunkiem. Taką własność informacji autor nazywa neutralnością typologiczną.

Informacja w ujęciu Floridiego posiada również inne cechy, które nazywa on „neutralnościami". Przez neutralność taksonomiczną autor rozumie stwierdzenie, że „dane" są bytami relacyjnymi – nic nie jest daną *per se*. I tak biała kartka papieru jest nie tylko tłem dla danej w postaci czarnej kropki, ale też częścią tej danej. Bycie daną jest własnością zewnętrzną. Nie ma zatem danych bez relacji, chociaż ogólna definicja informacji nie identyfikuje ich z żadną konkretną relacją.

Druga cecha informacji to neutralność ontologiczna rozumiana jako stwierdzenie, że nie ma informacji bez reprezentacji danych. Tezę tę można interpretować rozmaicie, a powtórzenie klasycznej debaty filozoficznej

realizm-antyrealizm prowadzić może do jej możliwych wyjaśnień lub uściśleń. Rozumiana materialistycznie, jako niemożność istnienia bezcielesnej informacji, wyraża się stwierdzeniem: nie ma informacji bez fizycznej reprezentacji. Takie założenie jest konieczne z punktu widzenia informatyki, która bierze pod uwagę fizyczne własności i ograniczenia nośników danych. Jednak teza ta dobrze pasuje również do niematerialistycznych koncepcji informacji. Przykładem jest monizm metafizyczny – którego wyrazicielem jest między innymi Wheeler – stwierdzenie, że istota wszechświata jest cyfrowa, składa się z informacji jako danych, a nie z materii czy energii, z obiektami fizycznymi jako jej złożonymi manifestacjami. Wheeler (1990) pisze:

> Wszystko z bitu (ang. *It from bit*). Innymi słowy, każde „coś" („*it*") – każda cząsteczka, każde pole siłowe, nawet kontinuum czasoprzestrzenne samo w sobie – wywodzi swoją funkcję, swoje znaczenie, istnienie (nawet jeśli w niektórych kontekstach pośrednio) z odpowiedzi na pytania typu tak-nie, wyborów binarnych, *bitów*. „It from bit" symbolizuje pogląd, że każdy obiekt fizycznego świata ma na dnie – w większości przypadków na bardzo głębokim dnie – niematerialne źródło i wyjaśnienie; to, co nazywamy rzeczywistością, powstaje ze stawiania pytań typu tak-nie oraz z notowania sprzętowo-wywołanych odpowiedzi; w skrócie – wszystkie fizyczne rzeczy mają pochodzenie informacyjno-teoretyczne i jest to wszechświat zaangażowany (ang. *participatory universe*) (s. 5).

Podobny pogląd głosi Wiener (1961): „Informacja jest informacją, nie materią czy energią. Żaden materializm, który tego nie przyznaje, nie może przetrwać w dzisiejszych czasach" (s. 132).

Trzeci rodzaj neutralności informacji, nazwany przez Floridiego (2005b) genetyczną, związany jest z analizą natury semantycznej danych. Zasada neutralności genetycznej stwierdza, że dobrze zdefiniowane dane mogą mieć znaczenie niezależnie od informowanego. Na przykład hieroglify egipskie uważano za informację nawet wtedy, gdy jeszcze nie potrafiono ich odczytać, to znaczy przed odnalezieniem Kamienia z Rosetty[30], który był kluczem do ich zrozumienia. Ogólna definicja informacji stwierdza zatem możliwość istnienia informacji bez podmiotu informowanego, gdyż znaczenie nie znajduje się w jego umyśle – a przynajmniej nie tylko w nim.

---

[30] Kamień ten odkryty został w czerwcu 1799 roku nieopodal miasta położonego w Delcie Nilu, zwanego w starożytności Rosetta, a obecnie al Raszid. Natrafili na niego francuscy saperzy z armii Napoleona, którzy mieli przygotowywać fundamenty pod budowę fortu. Jest to fragment steli z czarnego bazaltu (o wymiarach – wysokość: 118 cm, szerokość: 77 cm, grubość: 30 cm i waga: 762 kg) zachowany w stanie szczątkowym. Na kamieniu wyryty został tekst w trzech wersjach – po egipsku pismem hieroglificznym i demotycznym oraz po grecku.

Warto w tym miejscu odróżnić zasadę neutralności genetycznej od mocniejszej realistycznej tezy, głoszonej między innymi przez Dretske'a (1981), zgodnie z którą dane mogą mieć swoje własne znaczenie niezależnie od inteligentnego informującego. Typowy przykład takiej informacji to seria koncentrycznych pierścieni w przekroju drzewa, dzięki której można oszacować jego wiek. Informacja taka nazywana jest środowiskową.

Zastanówmy się teraz, czy ogólna definicja informacji, przedstawiona przez Floridiego, jest odpowiednia do interpretowania dwóch podstawowych typów informacji: instruktażowej oraz rzeczowej. Rozpocznijmy od typu bardzo ważnego z punktu informatyki – informacji instruktażowej (ang. *instructional information*), której przykłady znaleźć można między innymi w instrukcjach obsługi lub montażu. Są one podawane zazwyczaj w formie przepisu (procedury) wykonania pewnej czynności (na przykład zmontowania szafki) i mogą zawierać wyrażenia warunkowe typu: „jeśli…, to…, w przeciwnym przypadku…". Ten typ informacji nic nie mówi o sytuacji czy stanie rzeczy, w żaden sposób go nie modeluje, nie przedstawia, jest raczej wskazaniem, jak ją spowodować, w jaki sposób osiągnąć pewien stan. Informacja instruktażowa pojawia się również w wielu innych kontekstach: od formułowania założeń („niech $x = 3$"), poprzez rozkazy („zamknij drzwi"), ruchy pionków w grach, partytury utworów muzycznych, aż po zapis programu komputerowego.

Floridi twierdzi, że informacja instruktażowa ma również swoją „semantyczną stronę" – może być (przynajmniej potencjalnie) znacząca, a więc jest informacją w sensie definicji (GDI). Co więcej, może być także powiązana z informacją rzeczową w kontekście performatywnym, na przykład – w programowaniu, gdy określa się typ zmiennej. Warto jednak pamiętać, że informacja instruktażowa nie ma wartości logicznej, ponieważ formułowanie założeń, rozkazy, instrukcje, reguły gier, partytury utworów muzycznych czy też programy komputerowe nie mogą być zaklasyfikowane jako prawdziwe czy fałszywe.

Rozważmy teraz, w kontekście teorii informacji silnie semantycznej, wartość logiczną informacji rzeczowej (ang. *factual information*). Jest to bowiem ten typ informacji, która mówi informowanemu „coś o czymś", na przykład gdzie znajduje się jakieś miejsce lub która jest godzina – czyli której prawdziwość (fałszywość) jest istotnym atrybutem. Wielu filozofów twierdzi, że definicja Floridiego nie jest odpowiednia do mówienia o informacji rzeczowej. Przyjęcie, że informację stanowią wszelkie znaczące i poprawnie sformułowane dane – niezależnie od tego, czy wyrażają one prawdę czy fałsz, czy też nie posiadają żadnej wartości logicznej – ma swoje bardzo kontrowersyjne konsekwencje.

Pierwszą z nich można wyrazić poprzez stwierdzenie, że informacja fałszywa jest prawomocnym rodzajem informacji semantycznej, a nie pseudoinformacją. Dretske (1981) natomiast twierdzi, że informacja fałszywa oraz informacja chybiona (ang. *miss-information*) nie są rodzajami informacji – podobnie jak kaczki przynęty i gumowe kaczki nie są rodzajami kaczek.

Po drugie, przyjmując definicję Floridiego, wszystkie tautologie (prawdy konieczne) można zaklasyfikować jako informacje, chociaż nie wnoszą one niczego nowego do opisu rzeczywistości. Jest to niezgodnie z zasadą głoszącą, że im mniej prawdopodobny komunikat, tym więcej niesie informacji.

Po trzecie, nie jest nadmiarowym wyrażenie „prawdą jest, że *p*", gdzie *p* jest zmienną, za którą możemy podstawić dowolną informację. Na przykład ze stwierdzenia „'Ziemia jest okrągła' to prawdziwa informacja" nie można wyeliminować słowa „prawdziwa" bez utraty znaczenia.

Przyjrzyjmy się teraz modyfikacji definicji (GDI) zaproponowanej przez Floridiego, która ma ją przystosować do mówienia o informacji rzeczowej. W swej pracy (Floridi, 2005c) autor wymienia kilka najczęściej przytaczanych argumentów na rzecz tezy, że informacja fałszywa jest rodzajem informacji – a nie pseudoinformacją; próbuje także je obalić. Floridi rozważa sytuację S: „na kolacji będzie dokładnie dwóch gości, a jeden z nich to wegetarianin" oraz informację FI: „(A) na kolacji będzie dokładnie trzech gości i (B) jeden z nich to wegetarianin".

- FI zawiera prawdziwą informację. Jednak w istocie tylko niektóre jej składniki są prawdziwe.
- FI pociąga prawdziwą informację. Jednak stwierdzenie „na obiedzie będzie więcej niż jeden gość" jest prawdziwą konsekwencją informacji prawdziwej (TI), a nie FI.
- FI może nieść prawdziwe informacje, choć czasami – wyłącznie „nie wprost". Mogą to jednak być jedynie pewne metainformacje, na przykład dotyczące braku wiarygodności źródła komunikatu.
- FI może wspierać procesy decyzyjne. Stwierdzenie to zakłada, że FI znajduje się w kontekście, w którym mamy wystarczającą ilość prawdziwych metainformacji, dotyczących na przykład marginesu błędu. Jednak FI można potraktować jako zdanie bardziej użyteczne niż inne zdania: fałszywe „na obiedzie będzie dokładnie jeden gość" czy też prawdziwe – „na obiedzie będzie mniej niż tysiąc gości". Sytuacja ta nie pokazuje jednak, że FI jest prawomocną informacją, ale jedynie że: (i) fałszywa informacja może być pragmatycznie interesująca oraz że: (ii) analiza zawartości semantycznej instancji informacji σ musi brać pod uwagę stopień przybliżenia σ do swojego odniesienia, zarówno w przypadku, gdy σ jest prawdziwa, jak i wtedy, gdy jest fałszywa.

– FI jest znacząca i ma tę samą strukturę logiczną, co prawomocna informacja. Jest to po prostu stwierdzenie mylące. FI może – co najwyżej – nieść pewne informacje pochodne, dotyczące na przykład jej źródła. Co więcej, instancja informacji σ nie jest informacją tylko dlatego, że jest interpretowalna.
– Jeśli FI nie jest informacją, to czym jest? Do odpowiedzi na to pytanie potrzebne byłoby wprowadzenie nowej kategorii poznawczej, poza informacją i wiedzą. Floridi proponuje nazwanie dobrze sformułowanych, znaczących, ale fałszywych danych terminem „informacja błędna" (ang. *misinformation*).

Podsumowując, autor stwierdza: „wydaje się, że nie ma dobrych powodów, by traktować informację fałszywą jako rodzaj informacji" (Floridi, 2005c, s. 364).

Nieporozumienie dotyczące natury informacji fałszywej może być związane z mylącymi analogiami. Przykładami takiej informacji są: fałszywe stwierdzenia oraz niepoprawne dane. Fałszywe stwierdzenie jest nadal stwierdzeniem, a niepoprawna dana jest nadal daną. Można zatem, przez analogię, stwierdzić, że fałszywa informacja jest nadal informacją, tyle tylko że taką, która ma wartość logiczną „fałsz".

Problematyczne wydaje się nieodróżnianie dwóch sposobów używania słowa „fałszywy": przypadkowego (ang. *attributive*) oraz predykatywnego[31]. Rozważmy, dla przykładu, dwa przymiotniki: „stary" oraz „dobry". „Stary policjant" to osoba, która jest policjantem i jest stara. Jest to użycie predykatywne przymiotnika „stary". „Dobry policjant" natomiast to nie: „dobry człowiek zatrudniony w policji", ale raczej ten, który wypełnia nienagannie wszystkie obowiązki policjanta. Słowo „dobry" modyfikuje znaczenie słowa „policjant" – jest to użycie przypadkowe. I tak na podstawie tego rozróżnienia można zbudować następujący test: jeśli przymiotnik używany jest w sposób przypadkowy, to jego złożenie z rzeczownikiem nie może być rozdzielone. Co więcej, użycie przypadkowe może być zarówno pozytywne, jak i negatywne. Rozważmy przykład „dobrego policjanta". Jego pozytywne użycie klasyfikuje *x* jako *y* (jako dobrego policjanta). Z kolei jego użycie negatywne neguje jedną lub więcej z cech koniecznych dla *x*, by być *y*. Na przykład określenie: „fałszywy posterunkowy" (użycie przypadkowe) nie jest określeniem szczególnego rodzaju posterunkowego, lecz stwierdza ono, że „posterunkowy" nim nie jest (użycie negatywne). Dobrymi przykładami mogą tu być również: „fałszywy banknot", „sfałszowany podpis" czy „fałszywy alarm".

[31] Rozróżnienie to znane było już logikom średniowiecznym.

Rozważmy teraz związek pomiędzy „fałszywym stwierdzeniem" a „fałszywą informacją". Gdy mówimy, że zdanie *p* (na przykład „Ziemia ma dwa księżyce") jest fałszywe, używamy słowa „fałszywe" w znaczeniu predykatywnym. Można to łatwo sprawdzić za pomocą opisanego powyżej testu. Z łatwością możemy rozdzielić nasze stwierdzenie na dwa: „*p* jest zdaniem" oraz: „*p* jest fałszywe". Sytuacja zmieni się, gdy opiszemy *p* jako fałszywą informację. Używamy wówczas słowa „fałszywy" w znaczeniu przypadkowym (podobnie jak w przykładzie z „posterunkowym"), jako przeczenie tego, że *p* można zaklasyfikować jako informację. Poddajmy wyrażenie „fałszywa informacja" testowi. Nie można stwierdzić, że *p* stanowi informację dotyczącą liczby księżyców krążących wokół Ziemi oraz że jest fałszem. Dla porównania stwierdzenie, że „σ jest cyfrową informacją", rozkłada się bez utraty znaczenia na: „σ jest informacją" oraz: „σ jest cyfrowa". A zatem „fałszywa informacja" nie jest rodzajem informacji, który posiada odpowiednią wartość logiczną – fałsz, lecz po prostu nie jest to wcale informacja.

Widzimy więc, że ogólna definicja informacji Floridiego, aby była odpowiednia również dla informacji rzeczowej, wymaga uzupełnienia. Poza stwierdzeniem, że informacja składa się z pewnej ilości poprawnie sformułowanych znaczących danych, trzeba dodać, że jest ona prawdziwa. Ogólna definicja informacji przyjmuje wtedy postać:

> (RSDI) σ jest instancją DOS informacji [deklaratywnej, obiektywnej i semantycznej] wtedy i tylko wtedy, gdy:
> 1. σ składa się z *n danych*, dla $n \geq 1$;
> 2. dane te są *poprawnie sformułowane* (wdf);
> 3. wdf są *znaczące* (mwdf = δ);
> 4. δ są *prawdziwe* [ang. *truthful*] (Floridi, 2005c, s. 336)[32].

Przeanalizujmy teraz określenie miary informacji semantycznej, rozpoczynając od ponownego rozważenia przykładu podanego przez Floridiego (2005b). Niech *w* oznacza sytuację: „dziś na kolacji będzie trzech gości". Przypuśćmy, że podano następujące informacje:

T) „dzisiaj na kolacji będą goście lub też ich nie będzie",

V) „dzisiaj na kolacji będą goście",

P) „dzisiaj na kolacji będzie trzech gości".

Stopień informatywności (ilość niesionej informacji) zdania T jest zerowy, ponieważ jest ono prawdziwe zarówno w sytuacji *w*, jak i w sytuacji ¬*w*. Najbardziej informatywne natomiast jest zdanie P, ponieważ jest ono najbardziej precyzyjne.

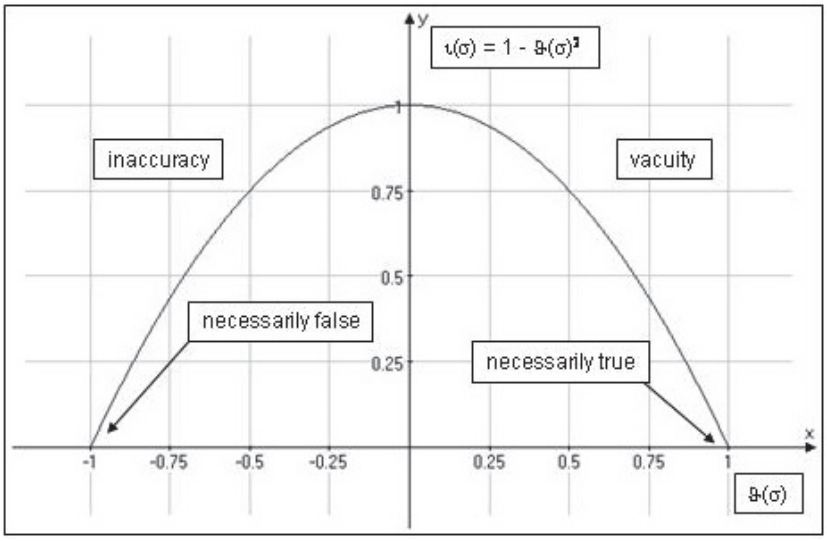
---

[32] Wtrącenia moje – I. B-K.

W ogólności, im większa odległość prawdziwego zdania σ od *w*, tym większa liczba sytuacji, do których ma ono zastosowanie i tym mniejszy stopień jego informatywności. Oznaczmy teraz przez ϑ odległość pomiędzy prawdziwym σ i *w*, którą przedstawiono na rysunku 4 na osi *x*. W powyższym przykładzie: $\vartheta(T)=1$, $\vartheta(P)=0$. Załóżmy, dla uproszczenia, że $\vartheta(V)=0{,}25$. Potrzebny będzie również wzór na obliczanie stopnia informatywności σ (oznaczanego literą ι) w relacji do $\vartheta(\sigma)$. Floridi twierdzi, że najlepszym (najbardziej eleganckim) rozwiązaniem jest zastosowanie dopełnienia do kwadratu wartości $\vartheta(\sigma)$:

$$\iota(\sigma)=1-\vartheta(\sigma)^2.$$

Gdy włączymy do odległości wartości ujemne dla fałszywego σ, to otrzymamy następujący wykres, (–1 to sprzeczność – kontrtautologia, 1 – tautologia):



Rys. 4. Stopień informatywności (źródło: Floridi, 2005b)

Z powyższego schematu możemy łatwo odczytać, że jeśli σ ma wysoki poziom informatywności, to zawiera dużą ilość informacji semantycznej, i na odwrót. Aby obliczyć ilość informacji semantycznej zawartej w σ w odniesieniu do ι(σ), musimy policzyć pole pod krzywą w przedziale [0,1].

Jak wiemy, największą ilość informacji semantycznej (oznaczaną przez α) niesie ze sobą zdanie P, którego $\vartheta = 0$. Odpowiada to polu pod całą krzywą

w dodatniej części układu współrzędnych. Mamy wtedy:

$$\alpha = \int_0^1 \iota(\sigma)dx = \frac{2}{3}$$

Przejdźmy teraz do zdania V z rozważanego przykładu. Można je traktować jako koniunkcję następujących zdań: „będzie 1 gość na kolacji", „będzie 2 gości na kolacji", ..., „będzie *n* gości na kolacji", gdzie *n* jest pewnym rozsądnym ograniczeniem, jakie chcemy rozważać. V zawiera zatem informacje nadmiarowe (być może wiele). Floridi określa „śmieci informacyjne" w zdaniu V jako „informację pustą" w nim zawartą. Ilość informacji pustej w V (oznaczana przez β) jest również funkcją odległości ϑ zdania V od *w*:

$$\beta = \int_0^{\vartheta} \iota(\sigma)dx$$

Ponieważ $\vartheta(V) = 0{,}25$, to otrzymujemy:

$$\int_0^{0{,}25} \iota(V)dx = 0{,}24479$$

Na rysunku 5. ilość pustej informacji w V pokazano jako zacieniony obszar.

Ilość informacji semantycznej w V można zatem zdefiniować jako różnicę pomiędzy α (maksymalną ilością informacji, jaką może nieść σ) oraz β (ilością pustej informacji niesionej przez σ). W ogólności, ilość informacji semantycznej γ w σ to[33]:

$$\gamma(\sigma) = (\alpha - \beta)$$

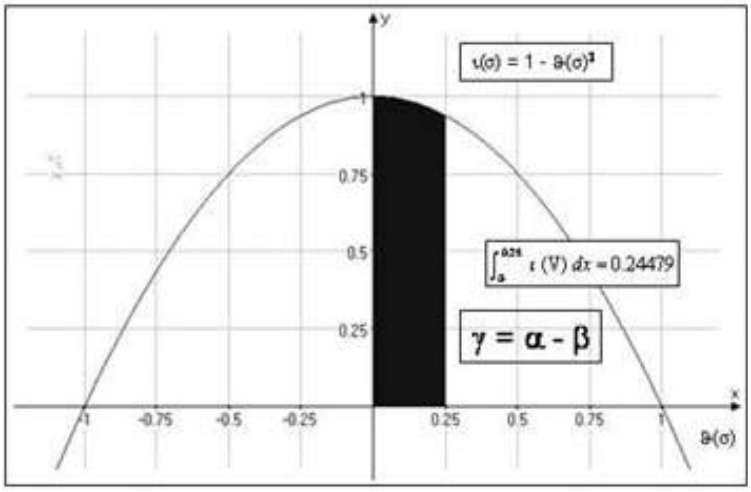
W przypadku zdania T (tautologii), α = β, a więc γ(*T*) = 0. T jest tak odległe od *w*, że niesie tylko informację pustą. Innymi słowy, T zawiera tyle samo informacji pustej, ile informacji semantycznej zawiera P.

---

[33] Jednostki informacji semantycznej autor mierzy w *sbitach*.

$$\int_0^1 \iota(\sigma)dx = 1\ sbit$$

Jeden sbit odpowiada maksymalnej ilości informacji semantycznej dotyczącej ustalonej sytuacji *w*, którą może nieść infon σ, gdy $\vartheta = 0$ (taką ilość informacji niesie zdanie P dokładnie opisujące sytuację *w*).



Rys. 5. Ilość informacji γ niesionej przez σ (źródło: Floridi, 2005b)

Tworząc przedstawioną powyżej teorię informacji silnie semantycznej, Floridi (2003) stawiał sobie trzy podstawowe cele:

> D.1 unikać nieintuicyjnych konsekwencji porównywalnych do paradoksu Bar-Hillela-Carnapa,
> D.2 traktować wartość logiczną σ nie jako dodatkową, ale jako konieczną cechę informacji, odpowiednią do analizy ilościowej […],
> D.3 rozszerzyć analizy ilościowe na całą rodzinę koncepcji związanych z informacją: pustość i niedokładność informacji, informatywność, informację błędną (nazywaną potocznie 'fałszywą informacją'), dezinformację (s. 11).

Na powyższej liście pierwszą pozycję zajmuje unikanie paradoksów, w tym paradoksu Bar-Hillela–Carnapa, rozumianego jako stwierdzenie, że najwięcej informacji niosą zdania sprzeczne (kontrtautologie). Sprawdźmy, czy Floridi cel ten osiągnął.

Sequoiah-Grayson (2007) analizuje sposób, w jaki teoria silnej informacji semantycznej unika paradoksu Bar-Hillela–Carnapa, używając wspomnianej wcześniej modalnej interpretacji treści semantycznej zdań, w której $\mathrm{Cont}(s)=_{def}\{x \in W: x \models \neg s\}$ (przez W autor oznacza zbiór wszystkich możliwych światów). Proponuje on przy tym przyjęcie wymogu przypadkowości w odniesieniu do informatywności, który formułuje następująco: „zdanie oznajmujące s jest informatywne ↔ s indywidualizuje przynajmniej niektóre, ale nie wszystkie w_i z W (gdzie $w_i \in W$)" (s. 338). Zatem aby zdanie s było informatywne, musi stwierdzać, który z możliwych światów jest światem

aktualnym. Ani tautologie, ani zdania sprzeczne takiego wymogu nie spełniają. Nie ma bowiem możliwości, by w naszym świecie zdania sprzeczne były prawdziwe czy też by tautologie były fałszywe.

Paradoks Bar-Hillela–Carnapa można jednak postrzegać inaczej, mianowicie w kontekście teorii informacji silnie semantycznej. Zauważmy, że ilość informacji semantycznej γ w infonie σ oblicza się przez odwołanie do pewnych własności prawdziwościowych i nie jest ona natychmiast identyfikowana ze wszystkimi informacjami, zagnieżdżonymi w σ, tak jak proponował to Dretske. Wyraźnie odróżnia to pojęcie ilości informacji semantycznej oznaczanej przez γ od tak samo nazywanego pojęcia oznaczanego przez Cont(σ). To drugie odnosi się bowiem do ilości informacji semantycznej związanej z σ *a priori*, na podstawie prawdopodobieństwa i niezależnie od stanu *w* (w jakim znajduje się obecnie rozważany system). Natomiast γ oznacza ilość informacji semantycznej związanej z σ również *a priori*, ale w kontekście pełnej informacji o systemie, określanej na podstawie wartości logicznej σ i jego stopnia rozbieżności z ustalonym stanem systemu *w*. Są to zatem dwa istotnie różne pojęcia, ujmujące informatywność z różnych perspektyw. Nie rozwiązuje to jednak do końca rozważanego paradoksu. Powstaje bowiem pytanie, co w takim razie oznacza Cont(σ).

Zdania sprzeczne lub kłamstwa nie mogą być w żaden niekontrowersyjny sposób rozumiane jako bardziej informatywne niż zdania prawdziwe. Czyż zatem Cont(σ) jest pojęciem bezużytecznym? By nadać mu właściwy sens, a tym samym uniknąć paradoksu Bar-Hillela–Carnapa, należy dokonać dwóch uściśleń: określić, co mierzy Cont(σ), oraz ograniczyć jego stosowalność.

Po pierwsze, Cont(σ) jest miarą nie informacji semantycznej, lecz – mówiąc ściślej – ilości danych w σ. Określa zatem zupełnie *niezinterpretowaną* informację, to znaczy dane, które nie są niesione przez σ, ale raczej tworzą (konstytuują) σ jako syntaktycznie poprawne kombinacje symboli lub sygnałów.

Po drugie, ponieważ Cont(σ) dotyczy jedynie danych i ich możliwych kombinacji *a priori*, to jej wartość nie może być wyznaczona dokładnie bez pewnych założeń dotyczących rozważanego systemu. Floridi (2003) formułuje następujące trzy warunki, które nazywa systemowymi:

> S.1) jednoznaczna indywidualizacja oraz dokładny opis danych systemu […];
> S.2) wygenerowanie całkowicie znormalizowanego opisu systemu danych jako zbioru wszystkich wzajemnie wykluczających się wiadomości, koniecznych i wystarczających do opisania w całości wszystkich możliwych stanów, w których może znajdować się system[34];

[34] Por. teoria informacji semantycznej – paragraf 3.2.4.

S.3) przypisanie prawdopodobieństwa p do każdego opisu stanu σ_i [...] w taki sposób, by spełnione były dwa warunki:

$$0 < p(\sigma_i) < 1 \text{ dla każdego } i$$

$$\sum_{i=1}^{n} p(\sigma_i) = 1$$

Dopiero po spełnieniu powyższych wymogów można przypisać dokładne znaczenie do Cont(σ) (s. 23).

Wartość prawdopodobieństwa jest określana wówczas zgodnie z S.3) w przedziale otwartym (0,1) nie dlatego, że nie jesteśmy w stanie przeanalizować prawdopodobieństwa σ całkowicie, ale dlatego, że jest to w danym przypadku niewłaściwe. Prawdopodobieństwo σ może być właściwie interpretowane jako miara ilości interpretowalnych danych w σ, tylko jeśli σ jest ciągiem poprawnie sformułowanych symboli lub sygnałów, o których *a priori* nie wiemy, czy są prawdziwe, czy fałszywe. Tylko wtedy możemy połączyć prawdopodobieństwo σ z liczbą danych w σ w tym sensie, że jedno jest wiarygodnym argumentem dla określenia drugiego.

Powyższe uściślenia Cont(σ) prowadzą do modyfikacji rozumienia związku pomiędzy informatywnością σ a jego prawdopodobieństwem. Gdy mówimy, że mniej prawdopodobne σ jest bardziej informatywne, czynimy pewne spostrzeżenie psychologiczne, dotyczące subiektywnych oczekiwań użytkownika, odnosimy się do stopnia rozbieżności σ z jednym lub więcej *w*. Im mniej bezsensowny jest infon σ, tym mniej możliwych światów go „wspiera" i tym bardziej informatywny staje się σ w odniesieniu do ustalonego *w*, który pełni funkcję punktu odniesienia. Informacja jest aktualną możliwością, która jest niezgodna z co najmniej jedną – ale nie ze wszystkimi innymi możliwościami. W teorii informacji silnie semantycznej zarówno sprzeczności, jak i tautologie są przypadkami granicznymi „nieinformacji" (ang. *uninformation*), brakiem zarówno pozytywnych informacji, jak i brakiem zaprzeczeń – błędnych.

Informacja w teorii Floridiego zawiera w sobie prawdziwość (ang. *information encapsulates truth*) – to znaczy, że prawdziwość jest warunkiem koniecznym dla zdania, by można je było zaklasyfikować jako informację. Takie rozumienie informacji, co prawda, eliminuje paradoks Bar-Hillela–Carnapa, budzi jednak wiele innych kontrowersji.

Fetzer (2004a) twierdzi, że definicja informacji Floridiego jest zbyt wąska, a jej przyjęcie zaciemnia rozróżnienie pomiędzy informacją, dezinformacją oraz informacją błędną[35]. Broni on „klasycznej koncepcji informacji",

[35] W wielu kontekstach istotne jest rozróżnienie kategorii informacji błędnej (ang. *misinformation*) i kategorii dezinformacji (ang. *disinformation*). Dezinformacja zawiera element

analizując jej trzy podstawowe konsekwencje, odrzucane przez Floridiego: (FI) informacja może być fałszywa, (TI) tautologie są informacjami, (NR) wyrażenie „prawdą jest, że …" nie jest nadmiarowe.

Standardowe, zdaniem Fetzera, podejście do informacji zakłada, że jest ona po prostu znaczącą (sensowną) daną, która może, ale nie musi być prawdziwa. Zdanie „Istnieje życie we wszechświecie poza Ziemią" jest sensowną daną, obecnie jednak nie wiemy, czy jest ona prawdziwa. Zdanie takie klasyfikuje się więc jako informację, która może być fałszywa. Zatem jeśli chcielibyśmy przyjąć koncepcję Floridiego, należałoby odróżnić „nformację" od „informacji". Powyższe zdanie można by wtedy traktować jako przykład nformacji, dla której, w odróżnieniu od informacji, prawdziwe są przynajmniej dwie z odrzuconych przez Floridiego tez: (FI) nformacja może być fałszywa, a (NR) stwierdzenie „prawdą jest, że…" nie jest nadmiarowe.

Fetzer (2004a) przedstawia następujące zestawienie koncepcji informacji (s. 224):

Tabela 7. Porównanie koncepcji Floridiego do zwykłej wiedzy

Koncepcja Floridiego	Zwykła wiedza
(1) dane	(1') przekonanie/sąd
(2) znaczące	(2') uzasadnione
(3) prawdziwe	(3') prawdziwe

W przypadku zwykłej wiedzy (w teorii nazywanej przez Fetzera standardową lub klasyczną) mówi się o sądzie (1'), że jest prawdziwy (3'), gdy istnieje pewne jego uzasadnienie (2'). Z kolei (1) dane nawet jeśli są (2) znaczące, to nie muszą być one (3) prawdziwe. Zatem (2') zawiera pewne wytyczne (warunek) dla (3') w przypadku zwykłej wiedzy, ale (2) nie daje ich dla (3) w przypadku koncepcji Floridiego.

Interesujący – w tym kontekście – jest status zdań będących tautologiami. Floridi, podobnie jak Bar-Hilell i Carnap, twierdzi, że tautologie nie są informacjami. Jednak zdanie właściwie zaklasyfikowane jako tautologia w języku L ma tę własność, że jego prawdziwość wynika z samego tego języka, niezależnie od innych czynników. A więc w przypadku tautologii sposób, w jaki spełniają one warunki (1) oraz (2), gwarantuje, że spełniają również (3), a więc tautologie są „jedynym rodzajem zdań, które *koniecznie*

---

intencji, który nie jest częścią znaczenia pojęcia „mylna (błędna) informacja". Pojęcie dezinformacji obejmuje w szczególności upowszechnianie niepełnej, niewłaściwej lub w jakikolwiek sposób mylącej informacji, w celu świadomego oszustwa.

spełniają warunek prawdziwości. Należą one do klasy znaczących danych, które muszą być prawdziwe!" (Fetzer, 2004a, s. 225).

Jak się wydaje, problem polega na rozróżnieniu zdań syntetycznych od analitycznych. W przypadku tych pierwszych spełnienie warunku (1) i (2) nie pociąga (3), natomiast zdania analityczne są jedynymi, które niejako „automatycznie" spełniają warunek prawdziwości (3). Skoro w teorii Floridiego informacja z definicji nie może być fałszywa czy niedokładna – to tautologie są informacjami.

Co więcej, jeśli przyjąć zasadę, że dla każdego zdania oznajmującego ono samo lub jego negacja muszą być fałszywe, to zdanie, które jest informacją, może mieć swoje zaprzeczenie, które nią nie jest. Jest to pewnego rodzaju paradoks. Fetzer (2004a) uważa, że „'[i]nformacja' nie powinna być ograniczana do zdań oznajmujących, które są prawdziwe, syntetyczne lub nie" (s. 226). Istnieje bowiem wiele przykładów znaczących danych, które trudno jest nazwać prawdziwymi, takie jak zdjęcia (w tym zdjęcia rentgenowskie), kręgi w pniu drzewa i inne. Również plamy krwi mogą być jedną z najważniejszych informacji znalezionych na miejscu zbrodni, a przecież trudno mówić o ich prawdziwości. Teoria informacji nieobejmująca takich przykładów nie może być poprawna.

Z przykładami wskazanymi przez Fetzera jako problematyczne dla teorii informacji Floridiego dobrze radzą sobie inne teorie. Pierwsza z nich to koncepcja semiotyczna, zgodnie z którą coś można zaklasyfikować jako informację, jeśli oznacza ona coś dla kogoś (czyni to informację relatywną do osoby, dla której coś znaczy). Koncepcja pragmatyczna natomiast zakłada, że coś jest prawdziwe, jeśli zawiera wytyczne właściwe dla osiągnięcia pewnych celów. Oczywiście to, czy znaczące dane takie właśnie są, czy też nie, jest relatywne, zależy bowiem od zdolności ich posiadaczy do wyciągnięcia korzyści z informacji niesionej przez te dane[36]. Fetzer (2004a) zestawił informację w koncepcjach: semiotycznej, pragmatycznej i w koncepcji Floridiego, w postaci tabeli (s. 227):

Tabela 8. Semiotyczna i pragmatyczna alternatywa dla koncepcji Floridiego

Pojęcie Floridiego	Pojęcie semiotyczne	Pojęcie pragmatyczne
Dane	coś	wskazówki
Znaczące	znaczy coś	dla kogoś
Prawdziwe	dla kogoś	by działać

[36] Szczegółowe omówienie problemu prawdziwości informacji oraz jej związków z wiedzą zawarł M. Hetmański w książce z roku 2013 *Epistemologia informacji*. Przedstawia on również epistemologiczne analizy informacji, które pomijamy.

Zarówno semiotyczne, jak i pragmatyczne pojęcie informacji czyni ją zależną od kontekstu – w tym sensie, że ilość informacji, jaka może być „wydobyta" z danych, zależy od środowiska, wiedzy i zdolności tych, którzy z nią obcują. Natura informacji jako znaczących danych wydaje się być wyjaśniona przez koncepcję semiotyczną, podczas gdy jej wpływ na działanie człowieka opisuje koncepcja pragmatyczna, analizująca również sposób, w jaki na postępowanie to może wpływać informacja fałszywa.

Podsumowując swoje stanowisko, Fetzer (2004a) stwierdza, że „istnieje wiele rodzajów znaczących danych, które nie mogą być ani prawdziwe, ani fałszywe, co w mojej ocenie pokazuje, ponad wszelką wątpliwość, że teoria informacji [Floridiego] jest całkowicie chybiona" (s. 229).

Istnieje wiele innych, oprócz opisanych powyżej, interpretacji informacji semantycznej. Jedną z nich jest ujęcie inferencyjne, definiujące informację w terminach przestrzeni wynikania (ang. *entailment space*) – zależy ona od poprawnego wynikania zrelatywizowanego do teorii danej osoby lub do stanu epistemicznego. Inną interpretację proponuje natomiast teoria systemowa, rozwijana w logice sytuacyjnej, w której zawartość informacyjna zdania *p* nie jest wyznaczona *a priori*, ale w terminach rzeczywistej zawartości, jaką niesie *p* w odniesieniu do danej sytuacji.

3.2.6. Teoria systemowa

David Israel i John Perry (1990, 1992) sformułowali definicję informacji wyrażoną w logice sytuacyjnej, obejmującą obiekty (na przykład zdjęcia rentgenowskie), które niosą (zawierają) pewne informacje, jednak nie można mówić o związanym z nimi prawdopodobieństwie. Szukają oni zatem innego niż w matematycznej teorii komunikacji sposobu definiowania informatywności komunikatów.

I tak „raportami informacyjnymi" autorzy nazywają zdania postaci: (1) Zdjęcie rentgenowskie wykazuje, że pies o imieniu Jackie ma złamaną nogę. (2) Fale akustyczne z głośnika niosą informację, że spiker powiedział „Nancy Reagan jest zirytowana". (3) Fakt, że zdjęcie rentgenowskie ma taką postać, wskazuje, że Jackie ma złamaną nogę. Kontekst informacyjny w tego rodzaju zdaniach tworzy czasownik informacyjny lub cała fraza („pokazuje", „wskazuje", „niesie informację"), razem z poprzedzającą ją frazą rzeczownikową. Zdanie podrzędne natomiast to treść informacyjna (ang. *informational content*). Obiekt określony przez początkową frazę rzeczownikową w raportach typu (1) i (2) nazywany jest nośnikiem informacji (na przykład „zdjęcie rentgenowskie", „fale akustyczne"), a fakt określony w raportach typu (3) – faktem wskazującym (ang. *indicating fact*).

Jeżeli dany raport informacyjny jest prawdziwy, to również prawdziwa jest jego zawartość informacyjna, to znaczy w szczególności, że jeśli zdanie: „Zdjęcie rentgenowskie wskazuje, że Jackie ma złamaną nogę" jest prawdziwe, to tak właśnie jest – Jackie ma złamaną nogę. Raporty informacyjne różnią się zatem od stwierdzeń dotyczących możliwości, przypominają natomiast zdania mówiące o tym, co konieczne.

Przed wprowadzeniem ścisłych definicji Israel i Perry (1990) wskazują na szereg zasad związanych z intuicyjnym rozumieniem pojęć „faktu" oraz „sytuacji", które ma obejmować stworzona przez nich teoria.

„(A) Fakty niosą informację.

(B) Treść informacyjna faktu jest zdaniem prawdziwym" (s. 3).

Ze zjawiskiem informacji ściśle związana jest pewna własność rzeczywistości polegająca na tym, że to, co dzieje się w jednej jej części, ma wpływ na to, co dzieje się w innej – na mocy pewnych praw czy regularności nazywanych przez autorów ograniczeniami (ang. *constraints*). Powodują one, że jeden typ sytuacji pociąga za sobą inne. W takiej właśnie rzeczywistości sytuacje niosą informację. Co więcej, jeśli zaistnieje sytuacja jednego typu, to niesie ona informację, że istnieją sytuacje typów, które za sobą pociąga. Na przykład jeśli istnieje ograniczenie (prawo fizyczne), że obiekty pozostawione bez oparcia blisko powierzchni ziemi spadają, to fakt, iż konkretne jabłko pozostawiono bez oparcia przy powierzchni ziemi, niesie informację, że jabłko to spadnie. Zatem koncepcja ta używa pojęcia informacji niesionej przez fakty w odniesieniu do ograniczeń.

„(C) Informacje niesione przez fakty są zrelatywizowane do ograniczeń" (s. 3).

Struktura rzeczywistości nie wymaga, by istniały istotne związki pomiędzy wydarzeniami, a żadne z nich nie zawiera informacji jedynie z racji tkwiących w nim własności. Jeśli wydarzenie zostanie zanurzone w różne rodzaje światów, w których obowiązują różne ograniczenia, to może ono nieść zupełnie inną informację. Zatem:

(D) Informacja, jaką niesie fakt, nie jest tkwiącą w nim (wewnętrzną) własnością.

Informacja zazwyczaj dotyczy faktów opisujących stan rzeczy gdzieś indziej i kiedy indziej, co czyni ją użyteczną i ciekawą. Można zatem sformułować prawo:

(E) Treść informacyjna faktu może dotyczyć odległych rzeczy i sytuacji.

Informację zawartą w zdaniach postaci (3) Israel i Perry nazywają przyrostową (ang. *incremental*). Aby lepiej zrozumieć to pojęcie, wystarczy zastanowić się nad tym, jaką informację może odczytać weterynarz ze zdjęcia rentgenowskiego. Jeśli nie wie on, do którego psa należy zdjęcie, to samo zdjęcie wskazuje jedynie to, że pewien pies został prześwietlony i że ma

złamaną nogę. Mamy wówczas do czynienia z informacją czystą (ang. *pure*). Jeśli jednak weterynarz wie, że prześwietlany był Jackie, to wygląd zdjęcia wskazuje informację przyrostową: Jackie ma złamaną nogę. Zatem zdjęcie rentgenowskie niesie informację przyrostową powstającą przez uwzględnienie faktu, że jest to zdjęcie zrobione Jackie'emu. Fakt ten łączy wskazaną sytuację i konkretne obiekty, których informacja dotyczy, i dlatego nazywany jest faktem łączącym (ang. *connecting fact*).

(F) Treść informacyjna może być konkretna; zdania, które są treścią informacyjną, mogą dotyczyć obiektów, które nie są częścią faktu wskazującego.
(G) Fakty wskazujące zawierają informację tylko w odniesieniu do faktów łączących; informacja jest przyrostowa dla danych faktów (s. 4).

Rozważmy teraz następującą sytuację. Jeśli włożymy zdjęcie rentgenowskie do szuflady na miesiąc, to po miesiącu nadal widzimy na nim złamaną nogę Jackie'ego. Oczywiście wtedy niesie ono informację tylko o tym, że Jackie miał złamaną nogę wtedy, gdy zostało zrobione zdjęcie. Sytuacja ta dobrze ilustruje dwie istotne własności informacji. Po pierwsze, różne fakty mogą nieść tę samą informację. Przypuśćmy, że *t* jest czasem zrobienia zdjęcia, natomiast *t'* jest czasem po miesiącu. Fakty: „zdjęcie ma konkretny wygląd w chwili *t*" oraz „wygląda ono tak samo w czasie *t*'" są istotnie różne, ale niosą tę samą informację. Oczywiście mogą istnieć również inne fakty świadczące o stanie nogi Jackie'ego, jak na przykład notatka sporządzona w kartotece. Po drugie, przedstawiona sytuacja jest przykładem składowania informacji. Nośnik informacji jest przechowywany w szufladzie, z której informacja jest pobierana. Taki system przechowywania działa poprawnie, dopóki sposób przechowywania nośnika gwarantuje zachowanie własności wskazywania. Nośnik może zachowywać tę własność bardzo długo, ale żaden system przechowywania informacji nie działa wiecznie.

Wyobraźmy sobie teraz, że zrobiono kopię zdjęcia rentgenowskiego i że została ona wysłana do innego weterynarza. Kopia ta również wskazuje, że Jackie miał złamaną nogę. Jest to prosty przypadek przepływu (transmisji) informacji. Zatem:

(H) Wiele różnych faktów, włączając zmiany w obiektach, własnościach, relacjach i miejscu czasoprzestrzennym, może wskazywać jedną i tę samą treść informacyjną – relatywnie do takich lub innych ograniczeń.
(I) Informacja może być przechowywana i transmitowana w wielu formach.

Zdjęcie rentgenowskie w czasie *t* niesie informację, że Jackie ma złamaną lewą nogę. Czy jest w tym coś dobrego dla zdjęcia? Nic. Co dobrego daje to Jackie'emu? Wiele. Dzięki tej informacji można psa odpowiednio leczyć.

Istnieje zatem różnica pomiędzy niesieniem lub zawieraniem informacji a jej *posiadaniem*. Agent lub urządzenie posiada informację, że *P* tylko w przypadku, gdy jest on w stanie, w którym zarówno niesie informację, że *P*, jak i kontroluje zachowanie urządzenia w sposób właściwy dla *P*. Israel i Perry formułują w związku z tym zasadę:

(J) Posiadanie informacji jest dobre; stworzenia, których zachowanie jest kierowane lub kontrolowane przez informację, mają większe szanse odnieść sukces niż te, które nie są w ten sposób kierowane.

Wszystkie przedstawione powyżej cechy informacji semantycznej, nazywane zasadami, są wytycznymi dla definicji stworzonej przez Israela i Perry'ego w obrębie teorii sytuacji.

Podstawowym założeniem teorii sytuacji jest istnienie rzeczywistości, która składa się z konkretnych części, a nie z konkretnych alternatyw (jak zakładano w teorii informacji semantycznej Bar-Hillela i Carnapa). Wszystko, co istnieje, wszystko, co się wydarza, wszystko, co jest prawdziwe, ma pewien status ze względu na naturę rzeczywistości. Jej części nazywamy *sytuacjami*.

Kiedy myślimy lub mówimy o rzeczywistości, potrzebujemy pewnego sposobu jej analizowania, który nazywamy *systemem klasyfikacji i indywidualizacji*. Składa się on z dziedziny sytuacji, relacji, lokalizacji i indywiduów. Każda relacja *R* związana jest z wieloma rolami, na przykład relacja jedzenia wiąże się z rolami: jedzącego, jedzonego oraz miejsca jedzenia, które to role odgrywają obiekty różnych rodzajów. Jedzący to jakiegoś rodzaju organizm, jedzenie – obiekt fizyczny, a miejsce jedzenia – pewien obszar czasoprzestrzenny. Relacja wraz z przypisanymi do ról obiektami tworzy *kwestię* (ang. *issue*). Dla każdej kwestii istnieją dwie możliwości, które nazywamy *stanami rzeczy* (ang. *state of affairs*).

Rozważmy za Israelem i Perrym przykład związany ze wspomnianą relacją jedzenia. Niech Ronald Reagan będzie jedzącym, chleb jedzeniem, a Biały Dom miejscem (oznaczonym przez l). Istnieją wtedy dwa możliwe stany rzeczy:

<<Jedzenie, miejsce: l, jedzący: Ronald Reagan, jedzenie: chleb, 1>>

<<Jedzenie, miejsce: l, jedzący: Ronald Reagan, jedzenie: chleb, 0>>,

gdzie znajdujące się na ostatniej pozycji zero oznacza, że relacja dla podanych argumentów nie zachodzi, natomiast jeden – że zachodzi, to znaczy, że Ronald Reagan je chleb w Białym Domu.

Czy dany stan rzeczy zachodzi czy też nie, zależy od sytuacji. Relację pomiędzy sytuacją a stanem rzeczy nazywamy *wspieraniem* lub *czynieniem faktycznym* (rzeczowym) i oznaczamy symbolem ⊨. Zatem s ⊨ σ oznacza, że sytuacja s wspiera σ. Z kolei ⊨ σ oznacza, że σ jest faktyczny (ang. *factual*).

Do rozważanych relacji wprowadzamy pewne parametry, które odpowiadają indywiduom lub lokalizacjom. *Infonem* lub parametrycznym stanem rzeczy nazywać będziemy stan rzeczy postaci <<…, a , …>>. Funkcję częściową z dziedziny parametrów do konkretnych obiektów nazywamy *kotwicą*. Gdy *f* jest kotwicą, to <<…, a,…>>[*f*]=<<…, *f*[a],…>>. Kotwica *f* spełnia infon *i* względem sytuacji *s* wtedy i tylko wtedy, gdy s ⊨ *i*[*f*]. Zatem kotwica przekształca infon w stan rzeczy.

Przejdźmy teraz do pojęć kluczowych dla zdefiniowania informacji. Niech σ będzie stanem rzeczy, wtedy [s| s ⊨ σ] jest *typem sytuacji* wspierających σ. Gdy *i* jest infonem (parametrycznym stanem rzeczy), to [s| s ⊨ *i*] jest *typem parametrycznym*, natomiast *i* jest *infonem warunkowym* dla *T* (*cond*(*T*)). Sytuacja *s* jest typu parametrycznego *T* względem *f*, jeśli s ⊨ *i*[*f*], gdzie *i* jest infonem warunkowym dla *T* oraz *f* jest zdefiniowana na wszystkich parametrach infonu *i*. Poniżej typy parametryczne określane będą w skrócie słowem „typ".

Wspomniane wcześniej ograniczenia są stanami rzeczy, których argumentami są typy sytuacji. Proste pociąganie (ang. *simple involvement*) jest relacją binarną. Jeśli *T* pociąga *T'*, to dla każdej sytuacji typu *T* istnieje jedna sytuacja typu *T'*. Piszemy wtedy <<Pociąga, *T*, *T'*; 1>>. Z kolei relatywne pociąganie (ang. *relative involvement*) jest relacją ternarną. Jeśli *T* pociąga *T'* względem *T''*, to dla każdej pary sytuacji pierwszego i trzeciego typu istnieje sytuacja typu drugiego, co oznaczamy <<Pociąga$_R$, *T*, *T'*, *T''*; 1>>.

W powyższej teorii sytuacji[37] Israel i Perry (1990) formułują następującą definicję:

> Niech C będzie pewnym ograniczeniem. Fakt σ *niesie czystą informację, że P relatywnie w stosunku do* C wtedy i tylko wtedy, gdy:
> 1) C=<<Pociąga, *T*, *T'*; 1>>
> 2) Dla każdej kotwicy *f* takiej, że σ=*cond*(*T*)[*f*], *P* = zdanie mówiące, że ∃s'(s' ⊨ ∃ a_1, … , a_n(*cond*(*T'*)[*f*])) (s. 9-10).

W przypadku omawianego wcześniej zdjęcia rentgenowskiego: wskazuje ono, że istnieje pies, któremu zrobiono zdjęcie i który ma złamaną nogę.

By zdefiniować pojęcie informacji przyrostowej, potrzebna jest relacja pociągania relatywnego. Niech C będzie pewnym ograniczeniem relacyjnym. Fakt σ *niesie przyrostową informację, że P relatywnie w stosunku do* C *i faktu* σ' wtedy i tylko wtedy, gdy:

1) C =<<Pociąga$_R$, *T*, *T'*,*T''*; 1>>
2) Dla każdej kotwicy *f* takiej, że σ=*cond*(*T*)[*f*] oraz σ'=*cond*(*T''*)[*f*], P jest zdaniem mówiącym, że ∃s'(s' ⊨ ∃ a_1, … , a_n (*cond*(*T'*)[*f*])).

---

[37] Więcej szczegółów znaleźć można w pracy (Israel, Perry, 1990).

Faktem łączącym w przypadku zdjęcia rentgenowskiego jest to, że rozważane zdjęcie jest zdjęciem lewej nogi Jackie'ego. Ponieważ Jackie jest elementem (składową) tego faktu, jest on również argumentem wskazanego zdania mówiącego, że jego noga jest złamana.

Tak zdefiniowane pojęcia autorzy stosują do formalizacji informacji czystej i przyrostowej oraz do opisu jej przepływu w przypadku omawianego zdjęcia rentgenowskiego. Faktem wskazującym σ jest pewna postać zdjęcia w czasie *t*. Gdy mówimy o informacji czystej, mamy na myśli następujące proste ograniczenie: ilekroć istnieje stan rzeczy zawierający konkretną postać zdjęcia w czasie *t*, to istnieje też stan rzeczy zawierający nogę psa (będącą obiektem na tym zdjęciu) taką, która jest w tym czasie złamana. Zatem zdanie *P* mówi, że istnieje pies, którego zdjęcie rozważamy, oraz że ma on złamaną nogę. Informacja czysta dotyczy zdjęcia rentgenowskiego, a nie samego psa lub jego nogi. Ograniczenie to można przedstawić w sposób formalny, jak następuje:

$$T = [s|\ s \models \ll \text{zdjęcie}, x, t, 1 \gg \wedge \ll \text{ma} - \text{postać} - \Phi, x, t, 1 \gg]$$

$$T' = [s|\ s \models \text{jest} - \text{zdjęciem} - \text{czegoś}, x, y, t, 1 \gg \wedge \ll \text{ma} - \text{złamaną} - \text{nogę}, y, t, 1 \gg]$$

$$C = \ll \text{Pociąga}, T, T'; 1 \gg$$

Sytuacją wskazującą σ jest wyrażenie $\ll$ zdjęcie, a, t′, 1 $\gg \wedge \ll$ ma − postać − Φ, a, t′, 1 $\gg$, gdzie a jest zdjęciem rentgenowskim, natomiast t' oznacza czas. Zakładamy, że σ jest faktyczny, to znaczy $\exists s$ ($s \models \sigma$).

Niech teraz *f* będzie kotwicą zdefiniowaną w następujący sposób:

$$\sigma = cond(T)[f] = \ll \text{zdjęcie}, x, t, 1 \gg \wedge \ll \text{ma} - \text{postać} - \Phi, x, t, 1 \gg [f],$$

to znaczy *f*(x)=a oraz *f*(t)=t'.

Wtedy zdanie P ma postać:

$$\exists s'(s' \models \exists y \ll \text{jest} - \text{zdjęciem} - \text{czegoś}, x, y, t, 1 \gg \wedge \ll \text{ma} - \text{złamaną} - \text{nogę}, y, t, 1 \gg)[f].$$

Stwierdza ono, że stan rzeczy, który zawiera psa, będącego w czasie *t* obiektem *a* (rozważanego zdjęcia), który ma (w tym samym czasie) złamaną nogę – jest stanem faktycznym. Inaczej – jest to zdanie mówiące, że istnieje pies, którego nogę przedstawia zdjęcie *a* w czasie *t'* i którego noga jest złamana w czasie *t'*.

W podobny sposób Israel i Perry definiują informację przyrostową oraz przepływ informacji.

Ważnym aspektem informacji, który nie był poruszany w przypadku prezentowanych wcześniej teorii, jest jej użyteczność. Israel i Perry, jak już

wspomniano, odróżniają niesienie lub zawieranie informacji od jej *posiadania*. Według nich informacja, jaką niesie obiekt z tytułu bycia w pewnym stanie, nie jest, z natury rzeczy, związana z obiektem będącym w tym stanie. Zdjęcie rentgenowskie będące w stanie Φ niesie informację, że noga Jackie'ego jest złamana tylko w stosunku do określonych ograniczeń oraz faktów. W stosunku do innych ograniczeń i faktów ten sam stan może nieść inną informację.

Aby lepiej wyjaśnić posiadanie informacji, rozważmy przykład ostrzenia ołówków za pomocą elektrycznej temperówki. Gdy wkładamy do niej ołówek, naciskana jest dźwignia, która zamyka obwód, tym samym uruchamiając silnik obracający ostrze; następuje ostrzenie ołówka. W tym przypadku włożenie ołówka powoduje, że temperówka znajduje się w pewnym stanie, który niesie informację. W zwykłych okolicznościach stan ten pojawia się tylko po włożeniu ołówka – powoduje on pewne zachowania zachodzące wewnątrz temperówki. Zatem naciśnięcie dźwigni spełnia dwa zadania: niesie informację na temat stanu, w jakim znalazł się system – ołówek znajduje się w temperówce – oraz powoduje, że w systemie tym coś się dzieje.

Warto zauważyć, że możemy stwierdzić, iż elektryczna temperówka działa w odpowiedni sposób tylko wtedy, gdy znamy cel jej działania – ostrzenie ołówków. W stosunku do innego celu, na przykład denerwowania użytkowników, nie działa ona właściwie, ostrząc ołówek. Kiedykolwiek mówimy o sukcesie, porażce lub stosowności akcji, mamy na myśli pewien cel lub miarę sukcesu.

Przyjrzyjmy się teraz temu, w jaki sposób Israel i Perry łączą informację z jej użytecznością (służeniem pewnemu celowi). Załóżmy, że:

1. *G* jest celem – ostrzeniem ołówków.
2. *Naciśnięta-dźwignia*, *zamknięty-obwód* oraz *obracanie-ostrza* są stanami systemu, który jest rodzaju *K*.
3. Istnieje ograniczenie $C_{\text{czysta-inf}}$: jeśli system *a* rodzaju *K* jest w stanie *naciśnięta-dźwignia* w miejscu *l*, to istnieje ołówek umieszczony w *a* w miejscu *l*.
4. Istnieje ograniczenie C_K kierujące wewnętrznym działaniem systemu: jeśli *a* jest w stanie *naciśnięta-dźwignia* w *l*, to *a* przechodzi w stan *zamknięty-obwód* i później w stan *obracanie-ostrza*.
5. Istnieje ograniczenie $C_{\text{czysty-rez}}$: jeśli *a* jest w stanie *obracanie-ostrza*, to jeśli ołówek styka się z ostrzami systemu *a*, to jest on ostrzony.

Temperówkę (rodzaju *K*) zaprojektowano w taki sposób, że stan niosący informację, iż ołówek został włożony, uruchamia łańcuch zdarzeń służących celowi, do którego została ona stworzona – ostrzeniu ołówków. Projekt ten jest odpowiedni tylko w środowisku, w którym naciśnięcie dźwigni niesie informację na temat włożenia ołówka oraz w którym ruch ostrza powoduje

jego ostrzenie. Mówimy wtedy, że system jest *dostrojony* do ograniczeń w stosunku do celu ostrzenia ołówków. Oczywiście w innych, bardziej złożonych przypadkach opisane powyżej związki mogą być bardziej skomplikowane, a wtedy relacja dostrojenia może okazać się niewystarczająca.

Podsumowując, David Israel i John Perry przedstawili szereg zasad, które powinna uwzględniać teoria informacji semantycznej, nakreślili jej ramy w obrębie teorii sytuacji oraz zastosowali je do analizy działania agenta, uwzględniając jednocześnie użyteczność posiadania informacji. Podjęli oni również próbę rozwiązania pozornego konfliktu pomiędzy pojęciem informacji związanym z zewnętrznymi ograniczeniami i faktami a pojęciem agenta posiadającego informację i posługującego się nią w swoich działaniach.

3.2.7. Informacja jako wynik procesu

Ciekawą próbę zdefiniowania informacji w terminach niezależnych od dyscypliny, w której pojęcie to występuje, podjął Rober M. Losee (1990, 1997). Jako cel postawił on sobie stworzenie takiej definicji, która będzie odpowiednia dla wszystkich dziedzin nauki, a tym samym dla wszystkich naukowców posługujących się tym terminem – od fizyków po filozofów zajmujących się epistemologią.

Za punkt wyjścia Losee przyjął pojęcie procesu – w bardzo szerokim rozumieniu – zarówno jako prostych funkcji matematycznych, jak i jako procesów społecznych, których poziom złożoności wydaje się wykluczać możliwość dokładnego sformalizowania. Zakłada on przy tym, że procesy są wolne od błędów.

Autor uważa, że informacja ma cztery podstawowe cechy. Po pierwsze, *jest czymś*, choć jej natura nie jest jasna. Po drugie, powinna wprowadzać *nową* wiedzę, ponieważ powtarzanie już otrzymanej wiadomości nie jest informatywne. Po trzecie, jest prawdziwa – kłamstwo lub informacja nieoparta na faktach jest jedynie dezinformacją. I po czwarte, powinna ona mówić *o czymś*. Te właśnie cechy ma uwzględniać definicja informacji jako wyjścia procesu.

Liczni autorzy definicji informacji, z których wybrane przedstawiono powyżej, koncentrują się na różnych jej aspektach. Niektórzy zrównują ją ze znaczeniem, zauważając, że fakt usłyszenia pewnego zdania nie wystarczy, by uczynić to zdarzenie aktem informacyjnym; to właśnie znaczenie czyni zdanie informatywnym. Inni opisują ją w terminach wiedzy bądź też jako coś, co pojawia się w umyśle odbiorcy podczas przyswajania wiadomości. Jeszcze inni twierdzą, że to forma lub struktura systemu jest równoważna

informacji. Informacja w strukturze jest „niematerialnym duchem", który współistnieje z obiektem fizycznym, o którym informuje, a powstaje ona w procesie abstrakcji od materialnych aspektów rzeczywistości fizycznej. Zwolennicy matematycznego definiowania informacji zwracają natomiast uwagę na fakt, że jej odebranie powoduje eliminację niepewności, zmniejszenie niewiedzy bądź też usunięcie braku informacji o strukturze.

Losee uważa, że ogólna definicja informacji nie może pomijać żadnego z opisanych powyżej aspektów, co więcej, powinna uwzględnić dwa, jak się wydaje, trudne do pogodzenia wymagania: musi ona zachowywać jak najwięcej z potocznego rozumienia informacji oraz powinna być możliwie precyzyjna, by pozwalała na wyeliminowanie z jej zakresu w sposób jednoznaczny zjawisk z nią niezwiązanych.

Losee (1997) proponuje zatem następującą definicję: „[…] informacja jest produkowana przez wszystkie procesy i są to wartości charakterystyk wyjść procesów, które są informacją" (s. 5). Twierdzi przy tym, że definicja ta ujmuje większość koncepcji informacji występujących w różnych dziedzinach wiedzy. Przyjrzyjmy się teraz temu, w jaki sposób Losee uzasadnia jej adekwatność.

Rozpocznijmy od podanych wcześniej pożądanych cech ogólnej definicji. Informacja będąca składnikiem wyników procesów niesie zawsze „informację o czymś". Owo „bycie o czymś" związane jest z procesem lub funkcją zawierającą pewną reprezentację danych wejściowych (reprezentację „wejścia"), a ta z kolei może być wynikiem innej funkcji – niosąc informacje o jej wejściu, i tak dalej. Przykładem może być proces pieczenia ciasta, który rozpoczyna się przygotowaniem składników oraz zbioru instrukcji: pisanych, mówionych lub w pamięci kucharza. Postępując zgodnie z tymi instrukcjami, kucharz łączy składniki, a z powstałej masy po upieczeniu powstaje ciasto. Zbadanie go dostarcza informacji dotyczących zarówno samego procesu, jak i składników ciasta. Zatem proces pieczenia zmienia jeden zbiór składników („wejście" tego procesu) w inny zbiór – tworzący ciasto. Możemy więc mówić o procesie pieczenia jako o niosącym informację o oryginalnych składnikach.

Proces pieczenia jest niemal zawsze procesem nieodwracalnym, kucharz nie może „cofnąć się" od upieczonego ciasta do oryginalnych składników. Istnieją jednakże procesy całkowicie odwracalne – można wówczas cofnąć się od ich stanu końcowego do stanu początkowego. W takich procesach podczas ich wykonywania nie traci się żadnej informacji.

Losee twierdzi, że wszystkie procesy produkują informację, której zrozumienie wymaga poznania ich natury. Mając dostateczną ilość czasu i zasobów, można opisać wszystkie, nawet najbardziej skomplikowane procesy. Niektóre z nich można zdefiniować za pomocą pojęcia funkcji matema-

tycznej, która działając na argumencie (lub argumentach), daje pojedynczy wynik. Za pomocą funkcji probabilistycznych można przedstawić nawet procesy losowe, na przykład rzut monetą opisuje funkcja *f*(orzeł, ½), która daje wartość „orzeł" w przybliżeniu połowę razy. Procesy mogą być również opisywane jako algorytmy, zbiory reguł, zgodnie z którymi należy postępować w określonej kolejności, aby otrzymać wynik. Funkcję można również kojarzyć z programem komputerowym lub urządzeniem, które z charakterystyk wejścia (danych wejściowych) produkuje wynik o pewnej charakterystyce. Każdy proces może być zdefiniowany funkcyjnie jako jedna funkcja lub więcej[38].

Informacja pojawia się wówczas, gdy proces coś wytwarza. Niezależnie od sposobu jego opisu, zawsze można łączyć go z pewnymi zmianami w świecie – niesie on zatem pewną informację zarówno o sobie samym, jak i o swoim wejściu. Informacja jest zatem, według Loseego (1997), „wartością obecnie przypisaną do charakterystyki lub zmiennej, którą daje funkcja *f*(*x*) lub proces. Wartość, którą daje funkcja, niesie informację na temat argumentu funkcji *x* lub o funkcji *f*(), lub też o obu" (s. 10-11). Funkcja *f* jest odwzorowaniem z jednej dziedziny *X*, zbioru argumentów, w inną *Y* – zbiór możliwych wartości. System informacji to zbiór wartości produkowanych przez funkcję *f* nad dziedziną *X* (*Y*=*f*(*X*)). *X* jest dziedziną wiadomości, które mogą być transmitowane, natomiast *Y* – dziedziną otrzymanych wiadomości, będących informacjami o samym procesie i o zbiorze *X*.

Warto podkreślić, że zgodnie z powyższą definicją sam proces nie jest informacją. Również jego wejście nie jest informacją o danym procesie – chociaż może być informacją o innym procesie. Podobnie wyjście nie jest informacją samą w sobie – wyniki są informacjami tylko w tym sensie, że niosą informacje o procesie i jego wejściu.

Przejdźmy teraz do hierarchicznego modelu transmisji informacji zaproponowanego przez Loseego, który to model zdaniem autora daje możliwość opisywania informacji na wielu różnych poziomach.

Oznaczmy odbieranie fonemu (ang. *speech sound*) *x,* generowanego przez mówiącego przez funkcję phoneme(*x*), której wartości są informacjami produkowanymi przez słuchającego. Przez złożenie funkcji i ich odwrotności można modelować wiele procesów, włączając zachowania komunikacyjne wysokiego poziomu, takie jak sposób wysyłania i odbierania języka poprzez medium dźwiękowe. Informacja w języku jest transmitowana od nadawcy do odbiorcy poprzez kodowanie myśli w postaci dźwięków. Mamy zatem dwie formy reprezentacji: wyrażenie (ang. *phrase*) i dźwięk.

---

[38] Korelacja pomiędzy funkcjami, algorytmami i procesami związana jest blisko z tezą Churcha.

Proces ten można reprezentować poprzez następujące złożenie funkcji:

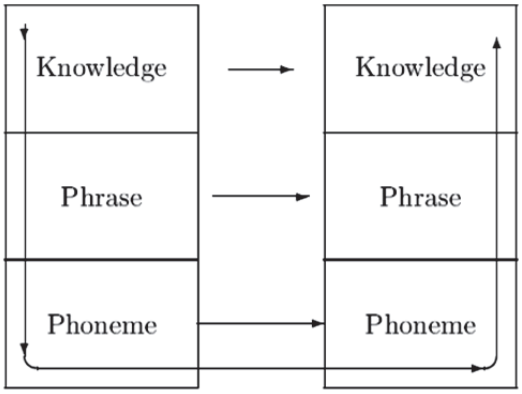
$$\text{phrase}(\text{phoneme}(\text{phoneme}^{-1}(\text{phrase}^{-1}(x)))),$$

gdzie phrase(x) jest wyrażeniem (zwrotem) odebranym przez słuchającego. W celu transmisji pewnego słowa koduje się je za pomocą funkcji phrase^{-1}, której wartość jest argumentem dla funkcji phoneme^{-1} umieszczającej zakodowany dźwięk w atmosferze. Z niej dźwięk jest następnie przechwytywany i dekodowany przez funkcję phoneme, dekodującą wiadomość do formy akceptowalnej dla funkcji phrase, zwracającej oryginalne słowo (jeśli oczywiście w opisanym procesie nie wystąpiły zakłócenia).

W celu dodania dodatkowej „warstwy" do opisanego modelu hierarchicznego konieczne jest wprowadzenie funkcji kodującej i dekodującej dla danego poziomu. Na przykład, „poziom wiedzy" można dodać, wprowadzając funkcje knowledge^{-1} oraz knowledge w następujący sposób:

$$\text{knowledge}(\text{phrase}(\text{phoneme}(\text{phoneme}^{-1}(\text{phrase}^{-1}(\text{knowledge}^{-1}(x)))))),$$

co Losee przedstawia graficznie za pomocą schematu (s. 14):



Rys. 6. Hierarchiczny model ludzkiej komunikacji. Strzałka w kształcie litery U reprezentuje przejście czegoś, co jest transmitowane, kodowane i dekodowane.

Możliwość rozbudowywania powyższego modelu o kolejne „warstwy" jest jedną z podstawowych jego zalet. Pozwala bowiem na mówienie o informacji na dowolnym poziomie ogólności, w zależności od zainteresowań i potrzeb jego użytkowników. Nie trzeba wtedy zastanawiać się nad naturą samej informacji, choć, oczywiście, inżynierowie i informatycy rozważają ją w inny sposób niż filozofowie czy kognitywiści. Jednak wszyscy oni będą mogli korzystać z zaprezentowanej hierarchii, rozbudowując ją o potrzebne

warstwy. Na dnie każdej takiej hierarchii znajduje się warstwa zawierająca mechanizm fizyczny umożliwiający proces komunikacji; jest to tak zwana „warstwa fizyczna". Informacja jest bowiem zawsze transmitowana poprzez serię procesów fizycznych.

Istnieje możliwość połączenia w całość wszystkich poziomów danej hierarchii poniżej konkretnego punktu. W ten sposób powstaje kanał, który Losse nazywa: „informacją jako rzeczą", co wskazuje na to, iż traktuje on informację jako obiekt.

W modelu hierarchicznym informacja jest niezależna od człowieka (jej odbiorcy). Upadek drzewa w lesie powoduje transmisję informacji – nawet jeśli nikt nie słyszy dźwięku gałęzi uderzających o ziemię. Powstały bowiem pewne zmiany w środowisku, i to pomimo tego, że nie mają one żadnego odzwierciedlenia w ludzkim uchu czy umyśle. Informacja została nadana, lecz nikt jej nie odebrał. W przedstawionym powyżej aparacie pojęciowym informacja nie wymaga obecności człowieka, a nauka o informacji nie powinna być rozumiana jako dyscyplina antropocentryczna. Przedmiotem teorii informacji jest bowiem informacja, a nie człowiek.

Losee twierdzi, że pojęcia dotyczące informacji i jej miary zaproponowane przez Shannona (opisane w paragrafie 3.2.1) można interpretować w terminach funkcyjnych. Również model hierarchiczny jest w pewnym sensie generalizacją modelu komunikacyjnego Shannona. Oba przedstawiają kodowanie informacji, jej przesyłanie i dekodowanie z uwzględnieniem kanału komunikacyjnego. Różnią się jednak w kilku kwestiach. Shannon opisuje system komunikacyjny, natomiast bardziej ogólny model hierarchiczny może objąć dowolny proces powodujący zmiany w świecie. Model ten jest zatem szerszy niż model wprowadzony przez Shannona, gdyż za pomocą terminów funkcyjnych umożliwia opisanie nie tylko dowolnego systemu komunikacyjnego, ale również pojęć bardziej abstrakcyjnych, takich jak percepcja, obserwacja, sąd czy wiedza. Przyjrzyjmy się teraz temu, jak można pojmować wiedzę w terminach funkcyjnych.

Wiedza jest często opisywana jako „uzasadnione prawdziwe sądy", więc tym samym sądy prawdziwe, dla których nie ma potwierdzenia, nie tworzą wiedzy. Dla opisania wiedzy w terminach funkcyjnych konieczne jest zatem wprowadzenie pojęć „prawdziwości" i „potwierdzenia" – w sposób zgodny z przedstawionym powyżej modelem. Rozumiejąc sąd prawdziwy, zgodnie z korespondencyjną teorią prawdy, jako właściwie reprezentujący rzeczywistość – oraz jego potwierdzenie w terminach własności funkcji generującej ten sąd, Losee (1997) twierdzi, że: „[s]ąd jest 'potwierdzony' wtedy i tylko wtedy, gdy wejście funkcji jest *właściwie* reprezentowane przez jej wyjście" (s. 26). Taka definicja wiedzy wykracza poza pojęcia językowe, nie jest antropocentryczna, a tym samym jest obiektywna, łatwo ją badać.

Zauważmy, że wiedza w tym ujęciu jest informacją, która jest zarówno prawdziwa, jak i potwierdzona.

Jak zatem w rozważanym modelu zdefiniować pojęcia informacji fałszywej czy dezinformacji? Jeżeli to, co jest nadawane (transmitowane), nie jest odbierane jako wysłane – czyli $x \neq f^{-1}(f(x))$ dla jakiegoś wejścia x funkcji f – oznacza to, że w rozważanym procesie nastąpiła utrata informacji. W przypadku funkcji częściowej lub podatnej na błędy lepiej jest interpretować jej wynik jako dezinformację, to znaczy informację, która jest częściowo lub całkowicie fałszywa.

Podsumowując opisane powyżej funkcyjne ujęcie informacji, Losee (1997) stwierdza:

> Pomimo tego, że nauka informacyjna rozumiana jest przez niektórych jako nauka społeczna, a przez innych jako gałąź fizyki, pojedyncza, szeroka, niezależna od dyscypliny, definicja informacji może służyć jako baza dla ścisłego oraz globalnego jej widzenia. Przyjęcie takiego modelu upraszcza dyskusje na temat wielu zależnych od dyscypliny pojęć, takich jak wiedza czy sąd. Użycie modelu hierarchicznego pozwala nam skupić badania na fenomenie informacji we wszystkich możliwych jej wcieleniach (s. 28).

Przedstawione powyżej próby zdefiniowania informacji i uchwycenia jej natury pokazują bogactwo i różnorodność powstających w tym celu teorii. Termin „informacja" jest często wiązany z filozofią, a współcześnie mówi się i pisze wiele – i to w różnych kontekstach – o tak zwanej „filozofii informacji".

3.3. Filozofia informacji

Filozofia informacji to dyscyplina stosunkowo młoda, której genezy należy szukać w takich dziedzinach jak: teoria komunikacji, cybernetyka, sztuczna inteligencja oraz informatyka, a która swą „dojrzałość" osiągnęła w latach 80. XX wieku[39]. Współcześnie w programach studiów zarządzania informacją prawie zawsze figuruje przedmiot filozofia informacji. Powstaje zatem pytanie, czym owa „filozofia informacji" jest.

Niektórzy autorzy (Adriaans, van Benthem, 2008; Lenski, 2010) widzą ją jako dyscyplinę techniczną, z głębokimi korzeniami w historii filozofii i konsekwencjami z tego wynikającymi dla różnych dyscyplin, takich jak metodologia, epistemologia i etyka. Inni, jak Floridi, prezentują „filozofię

[39] Można jednak mówić o wkładzie do filozofii informacji również tych autorów, którzy żyli przed erą informatyzacji.

informacji" jako kompletnie nową dyscyplinę, zdolną do zrewolucjonizowania samej filozofii.

Luciano Floridi w serii swych prac (2002, 2004b, 2008) dokładnie określił filozofię informacji jako dyscyplinę. Nie tylko podał jej definicję i zakres zainteresowań, ale również zaproponował tematykę jej badań w przyszłości. Stworzył następującą definicję: „filozofia informacji $=_{DEF}$ dziedzina filozofii zajmująca się (a) krytycznym badaniem pojęciowej natury i podstawowych zasad informacji, włączając jej dynamikę i zastosowania, oraz – (b) szczegółowym omówieniem i zastosowaniem metod obliczeniowych oraz z zakresu teorii informacji do problemów filozoficznych" (Floridi, 2001, s. 137).

Zwrot „dynamika informacji" występujący w tej definicji odnosi się do:

i) tworzenia i modelowania środowisk informacyjnych, w tym ich własności systemowych, form interakcji i tym podobnych,
ii) cykli życia informacji, to znaczy ciągów różnych stanów i aktywności, przez jakie przechodzi informacja od jej pojawienia się aż po możliwe zaniknięcie[40],
iii) obliczeń, zarówno w sensie Turinga, jak i w kontekście szerszym.

Pierwsza część powyższej definicji określa filozofię informacji jako nową dziedzinę badań. Ma ona między innymi odpowiadać na pytanie, czym jest informacja. Jednakże autor zwraca uwagę na to, że rozważania prowadzone w jej obrębie nie powinny być mylone z teorią informacji (w szczególności – z omówioną powyżej matematyczną teorią komunikacji). Zadaniem filozofii informacji nie jest bowiem stworzenie jednej ogólnej definicji czy też teorii informacji, lecz raczej badanie całej rodziny teorii, które analizują, wyjaśniają i opisują rozmaite własności informacji, a w tym – jej przesyłanie, dynamikę i użyteczność. Ma ona analizować te zagadnienia z nią związane, które wpływają na inne pojęcia filozofii, takie jak: byt, wiedza, prawda, życie, znaczenie.

Druga część definicji pokazuje, że filozofia informacji jest nie tylko nową dziedziną badań, ale również że wprowadza nową metodologię, którą można wykorzystać w poszukiwaniu odpowiedzi na pytania filozoficzne. Floridi zauważa, że metody, pojęcia i narzędzia informatyczne i obliczeniowe są już od dawna i z powodzeniem stosowane do wielu zagadnień filozoficznych:

---

[40] Floridi twierdzi, że typowy „cykl życia" informacji ma następujące fazy: pojawienie się (odkrycie, zaprojektowanie etc.), przetwarzanie i zarządzanie (zbieranie, modyfikowanie, organizowanie, indeksowanie, filtrowanie, uaktualnianie, sortowanie, przechowywanie, przesyłanie, transmitowanie etc.) oraz używanie (monitorowanie, aktualizowanie, wyjaśnianie, podejmowanie decyzji, nauczanie etc.).

– by poszerzyć nasze rozumienie zdolności poznawczych i językowych człowieka i zwierząt,
– w rozważaniach dotyczących możliwości powstania sztucznych form inteligencji (filozofia sztucznej inteligencji),
– w analizach procesów obliczeniowych i dedukcyjnych (filozofia obliczeń, filozofia informatyki, logika sytuacyjna),
– w wyjaśnianiu zasad organizacji życia i działania (filozofia sztucznego życia, cybernetyka i teoria automatów, teoria decyzji, teoria gier),
– w nowym podejściu do modelowania systemów fizycznych i pojęciowych (ontologia formalna, teoria systemów informacyjnych, filozofia rzeczywistości wirtualnej),
– w rozważaniach etycznych (etyka komputerowa),
– w analizach zjawisk psychologicznych, antropologicznych i społecznych charakteryzujących społeczeństwo informacyjne.

We wszystkich wymienionych powyżej zagadnieniach stosuje się metody i pojęcia obliczeniowe, co dowodzi, że filozofia informacji unifikuje powyższe dziedziny i programy, wprowadzając wspólne dla nich ramy teoretyczne. Floridi twierdzi, że posiada ona najpotężniejszy słownik pojęciowy, jaki kiedykolwiek rozwinęła filozofia, a to oznacza, że każde zagadnienie może zostać przeformułowane na terminy informacyjne. Ta właśnie moc semantyczna stanowi jej olbrzymią zaletę metodologiczną. Można zatem mówić o nowym, bardzo wpływowym w filozofii paradygmacie, określanym mianem „filozofii informacyjnej" lub „filozofii obliczeniowej". Floridi (2002) uważa, że „[f]ilozofia informacji ryzykuje stanie się synonimem filozofii" (s. 140) oraz że:

> [...] przedstawiana jako przyszła *philosophia prima*, zarówno w sensie Arystotelesowskim prymatu jej obiektu – informacji – który według filozofii informacji jest podstawowym składnikiem w każdym środowisku, jak i Kartezjańsko-Kantowskim sensie prymatu jej metodologii i problemów, filozofia informacji dąży do zapewnienia najbardziej cennego, kompleksowego podejścia do badań filozoficznych (s. 141).

Floridi (2002) twierdzi, że zdefiniowana przez niego filozofia informacji jest dojrzałą dyscypliną naukową, ponieważ: „(a) reprezentuje autonomiczną dziedzinę (*unikalne tematy*); (b) umożliwia innowacyjne podejście do zarówno tradycyjnych, jak i nowych zagadnień filozoficznych (*oryginalne metodologie*); (c) może stanąć obok innych dyscyplin filozofii, oferując systematyczne podejście do podstaw pojęciowych informacji i społeczeństwa informacyjnego (*nowe teorie*)" (s. 124).

Przedstawia on również kilkanaście problemów otwartych, których rozwiązaniem ma zająć się filozofia informacji (Floridi, 2004b). Część z nich

dotyczy ontologii informacji – jej natury w trzech ujęciach: *jako* rzeczywistość (informacja środowiskowa), informacja *na temat* rzeczywistości (informacja semantyczna) oraz – *dla* rzeczywistości (jako zbiór instrukcji, na przykład informacja genetyczna). Jako oddzielne zagadnienia badań Floridi proponuje analizę dynamiki informacji (w tym jej przetwarzania i zarządzania) oraz jej relacji do świata naturalnego (przyrodniczego). Z rozważaniami nad naturą informacji związane jest pytanie o możliwość stworzenia jednorodnej teorii informacji, przy czym autor skłania się ku poglądowi, że mówiąc o informacji w różnych dyscyplinach wiedzy, mamy do czynienia prawdopodobnie z siecią wzajemnie nieredukowalnych pojęć, co uniemożliwia stworzenie takiej teorii.

Kolejną grupę stanowią problemy semantyczne. W jaki sposób dane nabierają znaczenia? Jak znaczący zbiór danych staje się prawdziwy? Czy informacja wyjaśnia prawdę i znaczenie?

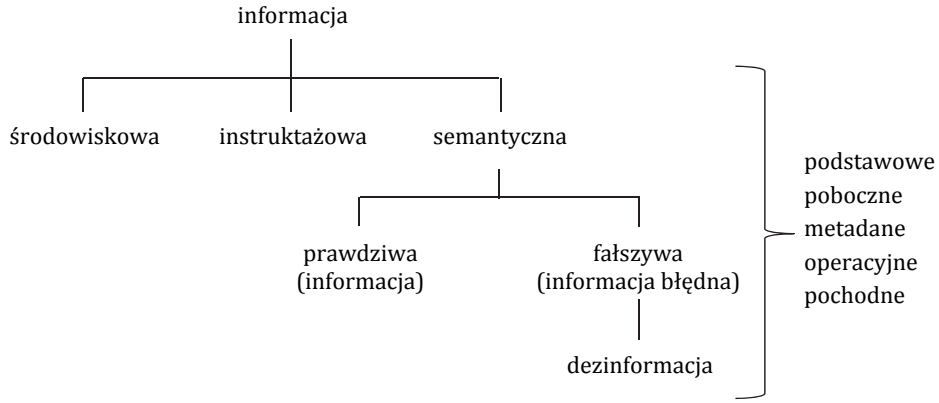
Ważnym zadaniem filozofii informacji jest zdaniem Floridiego szukanie odpowiedzi na pytania o charakterze filozoficznym, dotyczące natury ludzkiego poznania. Czy jest ono przetwarzaniem informacji? I co za tym idzie, czy inteligencję można analizować w terminach przetwarzania informacji? Czy możliwa jest sztuczna (czyli niebiologiczna) realizacja ludzkiej inteligencji? Istotną kwestią jest również szukanie nowych, opartych na podejściu informacyjnym rozwiązań klasycznych problemów filozoficznych, na przykład problemu natury ludzkiej wiedzy czy też relacji umysł–ciało.

Floridi (2008) pisze: „PI [filozofia informacji] zapowiada się jako jedna z najbardziej ekscytujących i owocnych dziedzin badań filozoficznych naszych czasów" (s. 19).

3.4. Podsumowanie

Przedstawione w tym rozdziale analizy informacji, choć nie wyczerpują nawet podstawowej problematyki z nią związanej, pokazują, że mamy do czynienia nie tyle z pojedynczą koncepcją, ile raczej z całym labiryntem pojęć i teorii je opisujących. Orientację w nim może ułatwić schemat przedstawiony przez Floridiego (2005a), ukazujący różne jej rodzaje.

Prócz rodzajów informacji na poniższym schemacie umieszczone zostały również nazwy pięciu podstawowych, zdaniem Floridiego, typów danych: podstawowe (ang. *primary*), poboczne (dodatkowe, ang. *secondary*), metadane, dane operacyjne oraz pochodne (ang. *derivative*); zostały one szczegółowo omówione w paragrafie 3.2.5. Z danych tych powstają różne rodzaje informacji.



Rys. 7. Taksonomia informacji

Informacja środowiskowa, którą można nazwać również naturalną, wpisana jest w otaczającą nas rzeczywistość przyrodniczą. Jest ona zawarta na przykład w pierścieniach wzrostowych drzewa czy też w odciskach palców. Rośliny (słoneczniki), zwierzęta (ameba), a nawet urządzenia (fotokomórka) mogą mieć zdolności do praktycznego wykorzystania takiej informacji, choć do jej istnienia nie jest konieczny żaden odbiorca.

Przykładów drugiego rodzaju informacji – instruktażowej – należy szukać w instrukcjach obsługi i montażu, regułach gier, partyturach utworów muzycznych oraz w zapisach programów komputerowych. Jest ona zazwyczaj przedstawiana w postaci przepisu (procedury) wykonania pewnej czynności i zawierać może instrukcje warunkowe (wyrażenia postaci „jeśli…, to…, w przeciwnym przypadku…"). Może również pojawiać się w innych kontekstach: w formułowaniu założeń („niech $x = 3$"), rozkazach („otwórz okno") czy opisach ruchów pionków w grach. Ten typ informacji, w przeciwieństwie do środowiskowej, nic nie mówi o sytuacji czy stanie rzeczy, w żaden sposób go nie przedstawia, jest raczej opisem sposobu, w jaki pewien stan można osiągnąć.

Informację semantyczną można rozumieć na wiele sposobów. Oto, jaką jej definicję znajdujemy w *Słowniku terminów i pojęć filozoficznych* (Podsiad, 2000): „Informacja semantyczna – funkcja określona na zdaniach (oznajmujących), pozostająca w następującym stosunku do ich prawdopodobieństwa: im zdanie jest mniej prawdopodobne na gruncie dotychczasowej wiedzy, tym więcej wnosi informacji, ponieważ jeśli okaże się prawdziwe – w większym stopniu wzbogaca wiedzę" (s. 390). Widzimy, że na rysunku 7 Floridi przedstawił dwa typy informacji semantycznej: prawdziwą (nazywając ją informacją) oraz fałszywą (którą opatrzył dopiskiem „informacja błędna",

ang. *misinformation*). Z tej drugiej kategorii wyróżnił z kolei dezinformację (ang. *disinformation*). Warto w tym miejscu zauważyć, że taki podział nie musi być – i nie jest – powszechnie uznawany za poprawny i jedyny możliwy. Nie ma bowiem zgody co do rodzajów i cech informacji. Istnieje wiele różnych teorii opisujących jej naturę (w tym jej związki z prawdziwością), a niektóre z nich zostały opisane w tym rozdziale.

Mówiąc o teorii informacji, informatycy mają zazwyczaj na myśli matematyczną teorię komunikacji Shannona, przedstawioną w paragrafie 3.2.1. Nie jest to jednak teoria informacji w zwykłym sensie, ponieważ rozpatruje ona tylko jej aspekt ilościowy, kładąc szczególny nacisk na mierzenie informacji zawartych w przesyłanych komunikatach. Shannon jest autorem dwóch powszechnie przyjmowanych idei: modelu komunikacyjnego (rysunek 3) oraz zasady mierzenia ilości informacji zawartej w komunikacie jako odwrotności jego prawdopodobieństwa (jest ona nazywana zasadą odwrotnej zależności). Czyli: im bardziej zaskakujący komunikat, tym więcej niesie informacji. Jeśli usłyszymy stwierdzenie, że jutro wzejdzie słońce, to nie jest ono zbyt zaskakujące i niesie niewiele informacji. Jeśli natomiast dowiadujemy się czegoś nowego, co jest dla nas zaskoczeniem (jest mało prawdopodobne), to, w pewnym sensie, jest bardziej informatywne.

Sposób mierzenia ilości informacji wprowadzony w matematycznej teorii komunikacji prowadzi do pewnej sprzecznej z intuicją własności, nazywanej w literaturze paradoksem małp. Zgodnie z tą teorią, największą ilość informacji niesie taki ciąg znaków (tekst), w którym prawdopodobieństwo wystąpienia każdej litery jest takie samo, to znaczy ciąg całkowicie losowy. Wówczas tekst pozbawiony treści, nonsensowny – na przykład pisany przez małpy na maszynie do pisania – niesie w sobie więcej informacji niż dzieła Szekspira. Co więcej, każde z tych dzieł zawiera mniej informacji niż tekst złożony z losowo pomieszanych tworzących go symboli (liter i znaków przestankowych). Jest to konsekwencja czysto syntaktycznego ujęcia informacji, całkowicie pomijającego między innymi jej znaczenie oraz kontekst. Teoria Shannona nie pozwala również na określenie ilości informacji zawartej na przykład w mapach lub zdjęciach.

Istnieje jednakże inny sposób mierzenia ilości informacji, niż za pomocą paradoksu małp, zaproponowany w latach 60. XX wieku, a nazywany obecnie algorytmiczną teorią informacji (por. paragraf 3.2.2). Informacja algorytmiczna[41] ciągu (na przykład złożonego z zer i jedynek) jest rozumiana jako długość najkrótszego algorytmu, który go generuje. Do jej matematycznego opisu Solomonoff (1960) wykorzystał pojęcie prawdopodobieństwa

[41] Informacja algorytmiczna jest znana pod wieloma różnymi nazwami, najczęściej jako „złożoność Kołmogorowa".

a priori skończonego ciągu symboli, jako wyznaczone przez najkrótsze wejście uniwersalnej maszyny Turinga, której wyjściem jest rozważany ciąg. Kołmogorow natomiast, dążąc do możliwości obliczenia informacji zawartych, na przykład w mapach, zaproponował rozważanie ilości informacji dostarczanej przez obiekt *y* (mapę) na temat obiektu *x* (regionu przedstawionego na tej mapie). Określił względną złożoność obiektu *y* z *x* jako minimalną długość 'programu' dla otrzymania *y* z *x*. Chaitin z kolei w swych rozważaniach wykorzystał pojęcie uniwersalnej maszyny Turinga. Chociaż każdy z autorów algorytmicznej teorii informacji – Solomonoff, Kołmogorow i Chaitin – miał inną motywację do jej stworzenia i wykorzystał inny formalizm do jej opisu, to jednak stworzone przez nich teorie są bardzo podobne. Wszystkie one przyjmują bowiem, że im krótszy program generujący ciąg, tym większe jego uporządkowanie i tym samym mniejsza złożoność (mniejsza ilość niesionych informacji).

Warto zauważyć, że korzystając z takiego ujęcia, możemy obliczyć ilość informacji zawartych nie tylko w mapach czy zdjęciach. Można bowiem to zrobić również dla obiektów, z którymi zazwyczaj nie wiążemy tego pojęcia – na przykład dla poszczególnych liczb (w tym liczby π), można bowiem określić długość najkrótszych programów je generujących.

Algorytmiczna teoria informacji, choć dobrze radzi sobie z niektórymi problemami teorii Shannona – w tym z paradoksem małp (unikając wykorzystania klasycznego pojęcia prawdopodobieństwa) oraz z obliczeniem ilości informacji w przypadkach map i zdjęć – nie wykracza jednak poza rozważania ilościowe. Podobnie rzecz się ma z jakościową teorią informacji Mariana Mazura.

Mazur stworzył koncepcję opartą na pojęciach wywodzących się z cybernetyki, wiążącą istotę informacji z procesami komunikacyjnymi i sterującymi, zachodzącymi w złożonym układzie. Nazwał ją jakościową teorią informacji (jej omówienie zawarto w paragrafie 3.2.3). Ograniczył swoje rozważania do zjawisk związanych ze sterowaniem rozumianym jako dążenie do pewnego celu. Nie jest to zatem teoria informacji samej w sobie – choć została zdefiniowana wprost, w terminach zamiany jednych komunikatów w inne.

Mazur przenalizował podstawowe sposoby informowania wiernego i wszelkie możliwe rodzaje zniekształceń informacji, zdefiniował także odpowiednik ilości informacji, nazywając go „liczbą informacji identyfikujących", którą można określić w niektórych, choć nie we wszystkich przypadkach kłopotliwych dla matematycznej teorii komunikacji (ze względu na to, iż oparta jest ona na prawdopodobieństwie). Można na przykład stwierdzić, jaką ilość informacji zawiera zależność pomiędzy długością średnicy koła a długością jego promienia.

Teoria Mazura traktuje informację, podobnie jak dwie omówione wcześniej teorie, jako wielkość fizyczną, ponieważ zmiany komunikatów to procesy fizyczne. Warto zwrócić uwagę, że występujący w nazwie tej teorii termin „jakościowa" nie odnosi się do informacji, lecz do samej teorii, a to może powodować nieporozumienia. Przedmiotem zainteresowań tej teorii jest określenie, czym w istocie informacja jest, jakie są jej rodzaje oraz w jaki sposób obliczać ilość informacji, a nie jej treść semantyczna. Takie ujęcie zazwyczaj odpowiada informatykom, koncentrującym swoją uwagę na przesyłaniu, przechowywaniu i ewentualnie na wyszukiwaniu informacji. Nie jest ono natomiast wystarczające dla filozofów, którzy poszukują odpowiedzi między innymi na następujące pytania: „Dlaczego traktujemy coś jako informację?", „W jaki sposób coś może nieść informację o czymś innym?", „Jaki jest związek informacji z prawdziwością, fałszem lub błędem?", „Kiedy informacja jest użyteczna?". Ilościowe teorie informacji nie odpowiadają na żadne z tych pytań, a w niektórych z nich unika się nawet prób definiowania informacji jako takiej. Jednak powstało – i nadal powstaje – wiele teorii ją opisujących, nie tylko na poziomie syntaktycznym, ale również semantycznym.

Jedną z pierwszych teorii informacji semantycznej, opisaną w paragrafie 3.4.2, stworzyli Bar-Hillel i Carnap. Zdefiniowali oni informację w terminach lingwistycznych, wykorzystując pojęcie prawdopodobieństwa indukcyjnego. Miarą treści semantycznej zawartej w p jest dopełnienie jego prawdopodobieństwa *a priori*: $\mathrm{Cont}(p) = 1 - \mathrm{P}(p)$. Takie probabilistyczne rozumienie treści informacji było później rozwijane, między innymi przez Hintikkę i Dretske'a.

Występujące w powyższym wzorze prawdopodobieństwo może mieć różne interpretacje. Bar-Hiller i Carnap zdefiniowali je poprzez konstrukcję formuł atomowych w wybranym języku formalnym. Dretske odniósł jej wartość do obserwowanego stanu rzeczy. Informację semantyczną można rozumieć również w terminach modalnych – treścią semantyczną zdania jest wówczas zbiór wszystkich możliwych światów, w których zdanie to jest fałszywe.

Teoria informacji semantycznej, niezależnie od interpretacji wykorzystywanego w niej prawdopodobieństwa, zachowuje wspomnianą wcześniej zasadę odwrotnej zależności, głoszącą, że ilość informacji niesionej przez dany komunikat jest odwrotnością jego prawdopodobieństwa. Wiąże się to z tak zwanym paradoksem Bar-Hillela–Carnapa: zdania sprzeczne (których prawdopodobieństwo wynosi zero) niosą najwięcej informacji. Odkrycie tej sprzecznej z intuicją własności doprowadziło do powstania wielu modyfikacji teorii informacji semantycznej. Najprostszym rozwiązaniem było wyeliminowanie *a priori* z rozważań zdań sprzecznych bądź też przyjęcie, że

wszystkie takie zdania niosą taką samą (zerową albo nieskończoną) ilość informacji. Jednak niektórzy filozofowie, aby uniknąć tego paradoksu, tworzyli nowe teorie opisujące informację semantyczną.

Jedną z nich, nazywaną teorią informacji silnie semantycznej (której analizę przedstawiono w paragrafie 3.2.5), zaproponował Luciano Floridi. Sformułował on ogólną definicję informacji semantycznej, według której informacja to zawartość semantyczna poprawnie sformułowanych, znaczących danych. Przez „poprawne sformułowanie" rozumie on zgodność ze składnią systemu, języka lub kodu, który analizujemy. Termin „znaczące" natomiast podkreśla, że dane muszą mieć pewne znaczenie w używanym systemie czy języku. Tak rozumiana informacja semantyczna posiada pewne cechy, które Floridi określa jako „neutralności". Przez neutralność taksonomiczną rozumie on stwierdzenie, że dane są bytami relacyjnymi – nic nie jest daną *per se*, bycie daną jest własnością zewnętrzną. Informacja nie zależy ani od typu tworzących ją danych, ani od swego nośnika – jest to neutralność typologiczna. Zasada neutralności genetycznej natomiast stwierdza, że dobrze zdefiniowane dane mogą mieć znaczenie niezależnie od informowanego, a zatem znaczenie znajduje się nie tylko w umyśle użytkownika. Inna cecha informacji – neutralność ontologiczna – to stwierdzenie, że nie ma informacji bez reprezentacji danych. Może być ona rozmaicie interpretowana, między innymi materialistycznie – nie ma informacji bez reprezentacji fizycznej – co wydaje się konieczne z punktu widzenia informatyki, która traktuje informację jako obiekt; jego pomiar, przesyłanie i przechowywanie jest jednym z podstawowych zadań informatyków.

Informacja silnie semantyczna ma również swoją miarę, zgodną z intuicyjnym jej rozumieniem. Zdanie niesie większą ilość informacji, gdy jest bardziej precyzyjne. Innymi słowy, im większa liczba sytuacji, do których ma ono zastosowanie, tym mniejszy stopień jego informatywności.

Floridi przedstawia również – często krytykowane – uzupełnienie swojej definicji informacji semantycznej o warunek jej prawdziwości, co ma uczynić ją bardziej odpowiednią do mówienia o informacji rzeczowej. Twierdzi on bowiem, że nie ma powodów, by uznawać informację fałszywą za rodzaj informacji.

Takie rozumienie pojęcia informacji krytykuje między innymi Fetzer (por. paragraf 3.2.5). Jego zdaniem jest ono zbyt wąskie i zaciemnia rozróżnienie pomiędzy informacją, dezinformacją oraz informacją błędną. Zwraca on również uwagę na to, że niektóre znaczące dane, takie jak zdjęcia, kod DNA czy też plamy krwi na miejscu zbrodni, niosą informację, choć trudno nazwać je prawdziwymi. Teoria informacji Floridiego nie jest zatem poprawna, ponieważ nie obejmuje wszystkich jej przykładów.

Lepiej z problemem tym radzą sobie inne teorie informacji: semiotyczna i pragmatyczna. Pierwsza z nich, zgodnie z którą coś można zaklasyfikować jako informację, gdy owo coś ma dla kogoś znaczenie, wydaje się dobrze wyjaśniać naturę informacji jako znaczących danych. Koncepcja pragmatyczna natomiast zwraca uwagę na wpływ informacji – również fałszywej, na zachowanie człowieka, koncentruje się ona bowiem na wpływie informacji na osiąganie pewnych celów.

Istnieją również inne interpretacje informacji semantycznej, w których można opisać informacje w przypadkach kłopotliwych dla teorii Floridiego. Jedną z nich jest teoria systemowa, przedstawiona w paragrafie 3.2.6, sformułowana w logice sytuacyjnej przez Davida Israela i Johna Perry'ego. Zakłada się w niej, że zawartość informacyjna zdania nie jest wyznaczona *a priori*, lecz w odniesieniu do danej sytuacji.

Israel i Perry przyjmują istnienie rzeczywistości składającej się z konkretnych części nazywanych sytuacjami, a wszystko, co w niej się wydarza, ma pewien status ze względu na jej naturę, między innymi na występujące w niej ograniczenia (na przykład prawa fizyczne).

Autorzy przedstawili dziesięć podstawowych cech informacji semantycznej, nazywanych przez nich zasadami i uwzględnionych w ich teorii: (a) fakty niosą informację, (b) treść informacyjna faktu jest zdaniem prawdziwym i (c) jest zrelatywizowana do ograniczeń, (d) informacja, jaką niesie fakt, nie jest tkwiącą w nim (wewnętrzną) własnością, (e) treść informacyjna faktu może dotyczyć odległych rzeczy i sytuacji, oraz (f) może ona być konkretna – zdania, które są treścią informacyjną, mogą dotyczyć obiektów, które nie są częścią faktu wskazującego, (g) fakty wskazujące zawierają informację tylko w odniesieniu do faktów łączących – informacja jest przyrostowa dla danych faktów, (h) wiele różnych faktów, włączając zmiany w obiektach, we własnościach, w relacjach i miejscu czasoprzestrzennym, może wskazywać jedną i tę samą treść informacyjną – relatywnie do takich lub innych ograniczeń, (i) informacja może być przechowywana i transmitowana w wielu formach, oraz (j) posiadanie informacji jest dobre – stworzenia, których zachowanie jest kierowane lub kontrolowane przez informację, mają większe szanse, by odnieść sukces, niż te, które jej nie posiadają[42].

Wykorzystując formalizm teorii sytuacji, Israel i Perry dokonali formalizacji pojęć informacji czystej i przyrostowej, ich przepływu oraz zastosowali je do analizy działania agenta, a uwzględniając użyteczność informacji, rozróżnili jej niesienie (lub zawieranie) od jej posiadania.

---

[42] Wyjaśnienie wszystkich pojawiających się na tej liście terminów znaleźć można w paragrafie 3.2.6.

Próbę stworzenia ogólnej definicji informacji ujmującej zarówno jej sens potoczny, jak i odpowiedniość dla wszystkich nauk, podjął Losee (por. paragraf 3.2.7). Określił on informację jako wyjście procesu, interpretowanego bardzo szeroko: od prostych funkcji matematycznych po złożone procesy społeczne. Losee twierdzi, że takie ujęcie informacji oddaje jej cztery podstawowe cechy: informacja jest „czymś" (choć jej natura nie jest jasna), wprowadza pewną nowość (powtarzanie tej samej wiadomości nie niesie informacji), jest prawdziwa oraz mówi „o czymś".

Losee wprowadził również hierarchiczny model transmisji informacji, który ma umożliwiać jej opisywanie na wielu różnych poziomach. Opiera się on na założeniu, że złożenie funkcji i ich odwrotności pozwala na modelowanie wielu procesów, w tym zachowań komunikacyjnych wysokiego poziomu (por. rys. 6). Model ten można w łatwy sposób rozbudowywać, dodając kolejne „warstwy" (poprzez zdefiniowanie dla nich odpowiednich funkcji kodujących i dekodujących), a to pozwala na mówienie o informacji na dowolnym poziomie ogólności, w zależności od zainteresowań i potrzeb jego użytkowników. W modelu hierarchicznym informację traktuje się jako niezależną od człowieka lub innego jej odbiorcy. Można w nim również zdefiniować w terminach funkcyjnych pojęcie „wiedzy".

Informacja, którą analizujemy w tym rozdziale, jest oczywiście blisko związana z wiedzą – czasami nawet jest z nią mylona. Szczegółowe omówienie związków tych pojęć wykracza poza ramy tej książki[43], warto jednak wspomnieć przynajmniej podstawowe kwestie.

Dunn (2008) tak pisze na temat związków pomiędzy informacją a wiedzą: „Lubię myśleć o informacji jako o tym, co zostaje z wiedzy, gdy odejmiemy uzasadnienie, prawdę, przekonania i inne składniki, takie jak wiarygodność, które odnoszą się do uzasadnienia. Informacja jest, tak jak była, czystą 'jałową myślą'" (s. 589).

Często przedstawia się związki pomiędzy pojęciem informacji a pojęciami pokrewnymi w postaci tak zwanej hierarchii DIKW, gdzie D oznacza niezinterpretowane (czyste) dane, I – informację, K – wiedzę (ang. *knowledge*), natomiast W – mądrość (ang. *wisdom*). Wydaje się jednak, że nie ma powszechnej zgody co do definicji elementów tej hierarchii. Niektórzy autorzy mówią o danych jako o symbolach lub liczbach, inni dopuszczają również obrazy lub dźwięki. Dane stają się informacją, gdy zostaną odpowiednio zinterpretowane i zapisane. Z kolei informacja nie staje się wiedzą w tradycyjnym sensie szeroko dyskutowanym przez filozofów, dopóki nie spełnia trzech testów Platona: uwierzone (zinternalizowane), uzasadnione,

[43] Dokładną analizę znaleźć można między innymi w książce *Epistemologia informacji* (Hetmański, 2013).

prawdziwe. Powszechne wykorzystanie komputerów do przesyłania, przechowywania i wyszukiwania informacji zmienia w pewien fundamentalny sposób klasyczne znaczenie tych terminów. Na przykład istnienie wyszukiwarek internetowych (takich jak Google) wymusza zmianę rozumienia roli eksperta. Jak wielu z nas ma pojęcie o sposobie działania Google? Jednak informacja – by można ją było nazwać wiedzą – musi zostać zinternalizowana i uzasadniona. W jaki sposób odbywają się te procesy w przypadku informacji znalezionych przez wyszukiwarki?

Związek pomiędzy informacją a wiedzą można jednak spostrzegać inaczej. Lenski (2010) twierdzi, że z punktu widzenia semiotyki informacja jest wymiarem pragmatycznym tego samego procesu, którego wymiarem semantycznym jest wiedza, a syntaktycznym – dane. Dzięki takiemu ujęciu można, jego zdaniem, analizować informację poprzez badanie wiedzy, a nie odwrotnie (jak dzieje się to we wspomnianej hierarchii DIKW). Wtedy kluczowym pojęciem jest wiedza, a nie informacja. Co więcej, autor stwierdza, że „informacja jest w gruncie rzeczy tym samym, co wiedza" (s. 111).

Zagadnienie związku pomiędzy informacją a wiedzą nie ma większego wpływu na rozumienie pojęcia „informacja" występującego w informatyce. Jak już wspomniano, informatycy zazwyczaj nie interesują się „naturą" informacji, lecz jej reprezentacją, mierzeniem i przesyłaniem.

Sposoby reprezentacji informacji pojawiającej się w informatyce można zobrazować w postaci kwadratu (Dunn, 2008), wzdłuż którego boków umieszczono kategorie: ustrukturyzowana/nieustrukturyzowana oraz tekst/ /multimedia. Jeszcze do niedawna, przed erą komputeryzacji, prawie cała informacja była tekstowa, natomiast obecnie szybko rośnie ilość informacji w postaci multimedialnej (filmy, zdjęcia etc.). Przykładem informacji ustrukturyzowanej są tradycyjne, tabelaryczne bazy danych. Z kolei informacje nieustrukturyzowane zawarte są w pajęczynie WWW (World Wide Web). Forma reprezentacji informacji ma znaczący wpływ na łatwość ich odnalezienia oraz operowania na nich[44]. W związku z tym czynione są próby „uporządkowania" Internetu, między innymi poprzez wprowadzenie struktury stron internetowych dzięki używaniu języków znacznikowych, takich jak HTML, a w szczególności XML. Nie ma jednak jednego, powszechnie przyjętego sposobu strukturyzowania informacji audio i wideo, a rosnące

[44] Widać to wyraźnie na przykładzie związku między sposobem zapisu liczb a wykonywaniem działań na nich. Wprowadzenie numeracji arabskiej w miejsce rzymskiego sposobu zapisu liczb uczyniło działania na liczbach łatwiejszymi. Bardzo trudno jest wykonywać działania bezpośrednio na liczbach zapisanych w systemie rzymskim (do obliczeń używano wtedy desek rachunkowych – abaków). Więcej na ten temat znaleźć można na przykład w (Bondecka-Krzykowska, 2012a) i *Historii powszechnej cyfr* (Ifrah, 2006).

znaczenie takiego typu informacji, nie tylko w Internecie, stawia w nowym świetle pytanie o naturę informacji i tworzących ją danych.

Pojęcie „informacji" pojawia się w informatyce w wielu różnych kontekstach, które nie doczekały się jeszcze pogłębionej refleksji filozoficznej. Przykładem może być pojęcie „informacji kwantowej". W klasycznym modelu komputera najbardziej podstawowe bloki informacji (bity) mogą być w jednym z dwóch odróżnialnych stanów oznaczanych przez „0" i „1". Bity kwantowe (kubity) mogą znajdować się w stanach „0" i „1", ale również w stanie pośrednim, będącym superpozycją zera i jedynki. W klasycznym rejestrze trzech bitów można zapisać jedną z liczb od 0 do 7, natomiast w rejestrze 3 kubitów – wszystkie liczby od 0 do 7 jednocześnie! Procesory operujące na rejestrach kubitów mogą wykonywać obliczenia, używając wszystkich możliwych wartości argumentów jednocześnie; jest to zjawisko nazywane czasami paralelizmem kwantowym. Dzięki niemu komputery kwantowe, wykorzystując specjalne algorytmy, są w stanie rozwiązywać klasyczne problemy w znacznie krótszym czasie niż komputery tradycyjne. Jednym z najbardziej znanych algorytmów kwantowych jest algorytm Shore'a[45] (stworzony przez Petera Shora z laboratoriów AT&T Bell) wykonywania faktoryzacji (rozkładu na czynniki pierwsze) dużych liczb, który widziany jest jako zagrożenie dla bezpieczeństwa tych metod szyfrowania danych, które opierają się na trudnościach z takim rozkładem.

Ciekawym filozoficznie aspektem obliczeń kwantowych jest ich odwracalność. Żadna informacja się nie gubi! Efekt taki można osiągnąć również w przypadku komputerów klasycznych, tyle tylko, że trzeba je specjalnie programować. W przypadku komputerów kwantowych odwracalność pojawia się sama, niejako „za darmo".

Inne zagadnienie filozoficzne pojawiające się w związku z informacją, a właściwie jej przechowywaniem, to interpretacja negacji. Tradycyjne bazy danych opierające się na logice dwuwartościowej działają w następujący sposób. Jeśli wylistujemy wszystkich pracowników danej firmy i na otrzymanej liście nie ma Jana Nowaka, to otrzymujemy informację, że nie jest on jej pracownikiem. Idea ta została zaimplementowana wprost w PROLOG-u, języku programowania opartym na fragmencie logiki pierwszego rzędu (tak zwanych klauzulach Horne'a) z dodaną negacją, interpretowaną jako „niepowodzenie". Można jednak traktować negację inaczej; na przykład Belnap (1977) zaprezentował ideę baz danych zawierających informację niespójną i użył ich jako przykładu konieczności wprowadzenia logiki czterowartościowej.

---

[45] Jest to, podobnie jak większość algorytmów kwantowych, algorytm probabilistyczny; zwraca on poprawną odpowiedź jedynie z pewnym prawdopodobieństwem.

Istnieje zapewne wiele innych, ważnych i ciekawych z filozoficznego punktu widzenia zagadnień informatyki, związanych zarówno z samym pojęciem „informacji", jak i z terminami z nią związanymi. Zaprezentowane w tym rozdziale analizy nie wyczerpują zagadnienia. Wynika to przede wszystkim ze złożoności i skomplikowania natury informacji jako takiej. Floridi (2008) pisze: „Informacja pozostaje pojęciem ulotnym (trudnym do zdefiniowania). Jest to skandal, zważywszy na fakt, że tak wiele podstawowych prac teoretycznych, zarówno w nauce, jak i w filozofii zależy od jasnego rozumienia natury informacji oraz pojęć pokrewnych" (s. 7).

Trudno zgodzić się z powszechnym używaniem terminu „informacja", nie tylko w dyskursach naukowych, ale również w życiu potocznym, w takich zwrotach jak „era informacji" czy „nauka informacyjna". Cóż bowiem znaczy używany w nich termin „informacja"? Czy jest to obiekt, coś, z czym mamy do czynienia na przykład w informatyce? Czy też jest to pojęcie ściśle związane ze znaczeniem i wiedzą? Wydaje się, że większość pochwalnych odniesień do „ery informacji" jest próbą ujęcia jej na dwa sposoby równocześnie, co prowadzi do licznych nieporozumień. Można zatem z całą stanowczością stwierdzić, że konieczne są dalsze badania, zarówno w obrębie informatyki, jak i filozofii, zmierzające do pełniejszego opisu natury informacji i związanych z nią zjawisk, zwłaszcza że „świat ujmowany jako czysta informacja nie tylko fascynuje nasze ciało i umysł, ale chwyta nas również za serce" (Heim, 1993, s. 283).

Rozdział 4

Rzeczywistość wirtualna[1]

Termin „wirtualny" jest dziś powszechnie używany. Mówi się o rzeczywistości wirtualnej gier komputerowych, o wirtualnych spotkaniach w sieci, wirtualnych spacerach po znanych miejscach, wirtualnych muzeach, wystawach, a nawet pieniądzach. Związane z nim wyrażenie „wirtualna rzeczywistość" używane jest do określania całej gamy zjawisk i aktywności człowieka związanych z komputerami. Obejmuje ono coraz szersze obszary nie tylko informatyki, ale również psychologii, socjologii i filozofii. W taki właśnie, bardzo szeroki sposób będziemy rozumieli rzeczywistość wirtualną w tym rozdziale – bez ograniczania się do konkretnej technologii, istniejącej lub wyimaginowanej. Konik (2009) tak pisze o wirtualności:

> Termin wirtualność współcześnie funkcjonuje przynajmniej na trzech poziomach: w znaczeniu informatyczno-technicznym (terminologia związana z technologią informacyjną, związaną z matematycznymi modelami cyfrowymi), filozoficznym (badającym struktury ontologiczne wirtualnych projektów jako światów możliwych, a także badającym problemy estetyczne i epistemologiczne), i ogólnym (w pojęciu ogólnym wirtualność oznacza to, co nierzeczywiste, nieistniejące, iluzoryczne, wyobrażone, możliwe, sztuczne, stworzone przez symulacje komputerowe [...] (s. 82-83).

Czym zatem jest rzeczywistość wirtualna? Jakie są jej cechy? Czy jest to zjawisko, które pojawiło się wraz z rozwojem technologii komputerowych i jest od nich całkowicie zależne? W rozdziale tym przedstawione zostaną

[1] Niektóre idee zawarte w tym rozdziale znaleźć można też w artykule pt. *Uwagi na temat ontologii wirtualnej rzeczywistości* (Bondecka-Krzykowska, 2012b)

próby odpowiedzi na powyższe pytania oraz inne, wybrane zagadnienia ontologii wirtualnej rzeczywistości.

Pojęcie „ontologia" (metafizyka), podobnie jak „wirtualna rzeczywistość", jest terminem bardzo szerokim. Przez „ontologię rzeczywistości wirtualnej" będziemy tutaj rozumieli filozoficzne badania nad opisem struktury wirtualnego świata. Wiążą się one z poszukiwaniem odpowiedzi na podstawowe pytania dotyczące statusu samej tej rzeczywistości (jakim rodzajem rzeczywistości jest świat wirtualny?, jaki jest status ontologiczny jego obiektów?), jak również na pytania dotyczące związków pomiędzy wirtualnością i realnością (czy rzeczywistość wirtualna to zmiana, rozszerzenie czy dodatek do rzeczywistości?, czy jest ona bardziej wirtualna, czy bardziej realna?). W obrębie tych badań znajduje się również analiza wpływu pojawienia się wirtualnej rzeczywistości na zmiany pewnych klasycznych koncepcji metafizyki oraz na powstanie nowych.

4.1. Wprowadzenie

Początków wirtualnej rzeczywistości można szukać w technologiach wytworzonych na potrzeby wojska oraz przemysłu rozrywkowego. Najwcześniejsze systemy rzeczywistości wirtualnej to symulatory lotów, używane przez armię amerykańską i NASA do szkolenia pilotów. Inne związane są z produkcją filmów, z tworzeniem na ich potrzeby coraz bardziej realistycznych obrazów i dźwięków. Czynnikiem, który szczególnie przyczynił się do ekspansji wirtualnej rzeczywistości, jest gwałtowny rozwój komputerów. Dodatkowe ich wyposażenie – na przykład okulary, hełmy i rękawice – oraz coraz lepsze karty graficzne ułatwiły nie tylko interakcję użytkowników z komputerami, ale również zagłębienie się w tworzony przez nie świat.

Warto zwrócić uwagę na to, że dla określenia zjawiska wirtualnej rzeczywistości, którego analizą zajmiemy się w tym rozdziale, używa się w literaturze różnych terminów, między innymi:

– wirtualna rzeczywistość,
– sztuczna rzeczywistość (termin zaproponowany przez Myrona Kruegera[2]),
– wirtualne środowiska (używany przez specjalistów z NASA i MIT),
– wirtualne światy (stosowany na uniwersytetach Karoliny Północnej i Waszyngtonu).

---

[2] Myron Krueger (ur. 1942 roku w Gary, Indiana, USA) – artysta, informatyk, autor serii artystycznych, komputerowych instalacji, które stały się zaczątkiem systemów rzeczywistości wirtualnej (na przykład CAVE), nazywanych środowiskami responsywnymi.

Istnieje również pojęcie „cyberprzestrzeń" wprowadzone przez Wiliama Gibsona w cyberpunkowej powieści *Neuromancer*, które odnosi się do „przestrzeni informacyjnej", „zespolenia informacji cyfrowej i ludzkiej percepcji" (Heim, 1993, s. 150).

Największą popularność spośród wymienionych powyżej terminów zyskało jednak określenie wprowadzone w latach 60. XX wieku – „wirtualna rzeczywistość" – a to między innymi dlatego, że było ono powszechnie używane przez producentów sprzętu i oprogramowania. Zapożyczone zostało przez Jarona Laniera[3] z dziedziny historii sztuki, gdzie mówi się o metaforze jako o „świecie wirtualnym". Jednak sam termin „świat wirtualny" pochodzi od Ivana Sutherlanda, który w roku 1968 opracował, prawdopodobnie pierwszy na świecie, system wirtualnej rzeczywistości, w którym używano nakładanych na głowę wyświetlaczy. Określał on świat wirtualny jako to, „co można zobaczyć w komputerowo wygenerowanym świecie, będąc przekonanym o rzeczywistym jego istnieniu" (za: Latawiec, 2009, s. 52). W dalszej części rozdziału terminów „rzeczywistość wirtualna" oraz „świat wirtualny" będziemy używali zamiennie, choć niektórzy badacze nie zgadzają się na takie utożsamienie[4].

Przyjrzyjmy się teraz definicjom zjawiska nazywanego „wirtualną rzeczywistością". Wydaje się, że powszechne rozumienie tego terminu dobrze oddaje Wikipedia: „Rzeczywistość wirtualna (ang. *virtual reality*) – obraz sztucznej rzeczywistości stworzony przy wykorzystaniu technologii informatycznej. Polega na multimedialnym kreowaniu komputerowej wizji przedmiotów, przestrzeni i zdarzeń. Może on reprezentować zarówno elementy świata realnego (symulacje komputerowe), jak i zupełnie fikcyjnego (gry komputerowe science-fiction)"[5]. Definicja ta koncentruje się na technologii używanej do tworzenia wirtualnej rzeczywistości i opisuje ją jako pewien obiekt – obraz świata rzeczywistego lub fikcyjnego.

Można jednak rozumieć rzeczywistość wirtualną szerzej, nie ograniczając się do technologii – na przykład Banse (2009) sądzi, że jest ona „możliwą (myślowo, konstrukcyjnie) rzeczywistością i wraz z tym określoną realnością" (s. 44).

Nie sposób jednak oddzielić wirtualnej rzeczywistości od jej użytkowników i wrażeń, jakich doświadczają oni podczas obcowania z nią. Dlatego też

---

[3] Jaron Zepel Lanier (ur. 3 maja 1960 roku w Nowym Jorku) – amerykański informatyk, kompozytor i futurysta. W 1984 roku założył firmę VPL Research, która wyprodukowała prototypowy sprzęt mający stać się elementem wirtualnej rzeczywistości, między innymi rękawice służące do manipulacji obiektami widocznymi na ekranie komputera.

[4] Por. np. (Latawiec, 2009).

[5] Wikipedia, http://pl.wikipedia.org/wiki/Rzeczywisto%C5%9B%C4%87_wirtualna.

definiuje się ją jako „model, do którego można wkraczać. Jest to interaktywny komputerowy system, który działa tak szybko i niezauważalnie, iż z jednej strony zapomina się o komputerze, z drugiej zaś – stwarzany jest świat, który postrzegany jest jak rzeczywistość" (Güner, 2002, za: Banse, 2008, s. 46-47).

W *Encyklopedii PWN* kładzie się jeszcze większy nacisk na wrażenia związane z przebywaniem w wirtualnym świecie: „*wirtualna rzeczywistość*, ang. virtual reality, VR, *inform.* doznania wzrokowe, słuchowe i dotykowe generowane za pomocą skomputeryzowanego sprzętu audiowizualnego i specjalnego oprogramowania; wykorzystywana zwłaszcza w celach rozrywkowych, także w badaniach naukowych"[6].

W literaturze znajdujemy wiele innych, często krańcowo różnych definicji rzeczywistości wirtualnej. Można je jednak zasadniczo podzielić na dwie grupy, co wynika z różnego rozumienia omawianego zjawiska. Definicje pierwszego rodzaju, podobnie jak ta cytowana wcześniej z Wikipedii, koncentrują się na technicznej stronie zjawiska; drugą grupę natomiast tworzą określenia podobne do wyżej cytowanego, pochodzącego z encyklopedii, i wskazują na pewne aspekty psychologiczne.

Największą wadą definicji „technicznych" jest ich nadmierna koncentracja na opisie określonego zestawu urządzeń i oprogramowania, dobieranego przy tym arbitralnie. Takie definicje mogą stawać się nieaktualne wraz z rozwojem technologii informatycznych, jak również nie dają możliwości porównania różnych systemów wirtualnych. Pomijają także całkowicie uczestnika zjawiska, który ma do czynienia nie tylko z aparaturą, ale przede wszystkim z jej wytworem – wirtualnym światem. Można zatem twierdzić, że rzeczywistość wirtualna jest w ogromnym stopniu zjawiskiem psychicznym lub psychologicznym, ponieważ dostarcza złudzenia przebywania w innym świecie.

Jednakże definicje drugiego typu, koncentrujące się tylko na psychologicznym aspekcie zjawiska, również nie są idealne. Często miewają one tak szeroki zakres, że obejmują nie tylko rzeczywistość wirtualną, ale również wszelkiego rodzaju odmienne stany świadomości, w szczególności te, które wywoływane są przez środki halucynogenne, sztukę czy praktyki medytacyjne. Kolejne, jeszcze inne określenia „rzeczywistości wirtualnej" łączą aspekt techniczny i psychologiczny, opisując ją jako sposób interakcji człowieka z komputerem (Aukstakalnis, Blatner, 1992).

Czym zatem jest tak różnie definiowane zjawisko, określane mianem „rzeczywistości wirtualnej"? Jakie są jego istotne cechy?

---

[6] http://encyklopedia.pwn.pl/haslo/3996681/wirtualna-rzeczywistosc.html.

4.2. Cechy wirtualnej rzeczywistości

Michael Heim w książce z roku 1993 *The Metaphysics of Virtual Reality* opisał podstawowe cechy wirtualnej rzeczywistości. Badacz twierdzi, że rzeczywistość wirtualna jest „zanurzającym, interaktywnym systemem, opartym na obliczalnej informacji. Te cechy definiujące sprowadzają się do 'trzech i': immersja, interaktywność i intensywność informacji[7]" (s. 6-7).

Co kryje się pod wspomnianymi „i"? Oznaczają one wybrane koncepcje związane z wirtualną rzeczywistością. Przeanalizujmy te oraz inne zjawiska, łączone z nią najczęściej – nie ograniczając się jedynie do „trzech i" – i spróbujmy utworzyć wyczerpującą listę cech ją definiujących. Zwolennicy różnych takich „list", jak pisze Heim (1993), „budują obozy, które z zapałem nie zgadzają się co do istoty tego, co stanowi rzeczywistość wirtualną" (s. 110). Przedstawione poniżej analizy mogą okazać się pomocne w określeniu cech zjawiska nazywanego wirtualną rzeczywistością.

4.2.1. Symulacja

Rzeczywistość wirtualną określa się często mianem „symulacji komputerowej". Aby zbadać trafność tego terminu, rozważmy najpierw pojęcie „symulacji" (ang. *simulation*). Można by zdefiniować ją jako „odtwarzanie własności obiektów rzeczywistych w środowisku cyfrowym" (Gurczyński, 2013, s. 125), przy czym „symulacja doskonała to taka, która nie pozwala się zorientować, że mamy do czynienia z symulacją" (tamże, s. 126). Warto zwrócić uwagę na to, iż symulacje obecnie nie tylko wiernie odzwierciedlają rzeczywistość, ale w wielu przypadkach nawet ją „przerastają". Zarówno statyczne obrazy graficzne, jak i dźwięki przewyższają jakością swoje analogowe odpowiedniki.

Pojęcie „symulacji" kojarzy się najczęściej z systemami nazywanymi „symulatorami" (na przykład symulatory lotu czy jazdy samochodem) oraz z grami komputerowymi. W pierwszym przypadku wierne odwzorowanie rzeczywistości jest wręcz niezbędne. Obecnie powszechne jest kształcenie pilotów na symulatorach lotów, a jego jakość zależy między innymi – choć nie tylko – od jakości symulacji. Podobnie rzecz ma się z symulacjami wykorzystywanymi w edukacji, na przykład z programami do nauki gry na instrumentach czy też gry w golfa.

[7] Ang. „immersion, interactivity and information intensity".

Z symulacją spotykamy się jednak i w innych przypadkach. Ikony reprezentujące foldery oraz pliki są również rodzajem symulacji – reprezentacją rzeczywistych obiektów. Są nią także zdjęcia, filmy czy niektóre obrazy. Oczywiście jeśli tak właśnie rozumie się pojęcie symulacji, to nie okaże się ona zjawiskiem nowym. Jeszcze przed erą wirtualnej rzeczywistości powstawały symulacje obiektów rzeczywistych, jak chociażby zdjęcia czy filmy.

Zauważmy jednak, że obiekty będące symulacjami bardzo często nie są nierzeczywiste. Usunięcie ikony reprezentującej plik dokumentu jest jednoznaczne ze zniszczeniem samego tego dokumentu. Jest to działanie zupełnie realne i ma swoje konsekwencje w rzeczywistości. Coraz więcej ludzkich działań przenosi się do rzeczywistości symulowanej, podczas gdy operowanie na jej obiektach ma swoje dokładne realne odpowiedniki – na przykład przelew dokonany w wirtualnym banku powoduje zmianę salda rzeczywistego konta.

W nauce i technice pojęcie symulacji ma dwa znaczenia (por. Latawiec, 2009). Oprócz wspomnianego już odwzorowania oryginału, pochodzącego zazwyczaj z rzeczywistości empirycznej, mówi się o symulacji jako o metodzie badawczej. Polega ona na sprawdzaniu lub odkrywaniu własności rzeczywistości z wykorzystaniem jej modelu w wirtualnym świecie. Aby sprawdzić pewne zjawisko lub proces, zazwyczaj niepoddające się weryfikacji bezpośrednio[8], formułuje się (tworzy) ich model, który poddawany jest sprawdzeniu w środowisku komputerowym (po napisaniu odpowiedniego programu, uruchomieniu go, weryfikacji i interpretacji jego wyników). Jest to rodzaj eksperymentu, w którym odkrywa się własności rzeczywistości, a nie świata wirtualnego (por. paragraf 2.2.2). Również w tym znaczeniu pojęcia „symulacja" znaczącą rolę odgrywa jej wierność oryginałowi (czyli dobre przybliżenie badanych procesów czy zjawisk).

Czy jednak pojęcie to wystarczy do zdefiniowania rzeczywistości wirtualnej? Zdecydowanie nie. Jeśli rozumiemy je szeroko, jako odzwierciedlenie cech obiektów rzeczywistych, to przykładami symulacji są zdjęcia i filmy, które nie mają wiele wspólnego z rzeczywistością wirtualną. Co więcej, nie wszystkie wirtualne światy są odzwierciedleniem światów rzeczywistych – wystarczy przyjrzeć się grom komputerowym, by znaleźć liczne przykłady „rzeczywistości wirtualnych", które nie mają niemal nic wspólnego z otaczającym nas światem. Ponadto do tworzenia symulacji nie są konieczne techniki cyfrowe, wystarczą obrazy analogowe lub modele czysto matematyczne. Nie można więc przyjąć, że symulacja definiuje wirtualną rzeczywistość.

Co zatem odróżnia te symulacje, które jesteśmy skłonni uznać za wirtualną rzeczywistość, od pozostałych? Jest to ich interaktywność.

---

[8] Na przykład ze względu na trudności lub niebezpieczeństwo związane z jego obserwacją, które pojawiają się w przypadku badań nad rozprzestrzenianiem się chorób zakaźnych.

4.2.2. Interaktywność

Gurczyński (2013) twierdzi, że „[i]nteraktywność jest podstawową i charakterystyczną cechą rzeczywistości wirtualnej. Wrażenie przebywania w innym niż rzeczywisty świecie powstaje w dużej mierze na skutek tego, że możemy wchodzić w interakcje z elementami środowiska wirtualnego" (s. 136). Interaktywność, rozumiana jako możliwość wzajemnego oddziaływania obiektu i jego użytkownika, jest często traktowana jako warunek konieczny wirtualności. Komputerowe reprezentacje tym różnią się od innych symulacji, na przykład od zdjęć czy modeli matematycznych, że ludzie mogą wchodzić z nimi w interakcje, które przypominają kontakt z prawdziwymi przedmiotami. Ludzie mogą również powodować, że komputerowe symulacje „robią" pewne rzeczy. Jest to ta cecha, której nie mają inne formy reprezentacji rzeczywistości.

Warto jednak pamiętać, że interakcja nie jest cechą przysługującą tylko wirtualnej rzeczywistości. Mamy z nią bowiem do czynienia również podczas wideokonferencji, w procesie nauczania na odległość (w tym w modnym obecnie e-learningu) bądź też w programach radiowych i telewizyjnych wykorzystujących na przykład łączność telefoniczną lub audiowizualną z widzami. Interakcja jest zatem zjawiskiem szerokim, występującym nie tylko w odniesieniu do technologii komputerowych. Przyjrzyjmy się zatem próbom jej zdefiniowania.

Definicja zaproponowana przez Steuera (1992) pozwala na porównywanie systemów ze względu na stopnień interaktywności, rozumianej jako sposób, w jaki użytkownik może uczestniczyć w modyfikowaniu danego systemu (środowiska) w czasie rzeczywistym. Definiuje on trzy elementy określające stopień interaktywności: szybkość, zakres oraz mapowanie.

Szybkość interakcji, którą Steuer rozumie czysto technicznie, w znaczący sposób wpływa na odczucie realności danego środowiska. Nawet systemy o gorszej grafice, ale działające szybko powodują, że użytkownicy mają wrażenie rzeczywistych działań. Każde opóźnienie reakcji na te działania zmniejsza odczucie realności. Dotyczy to szczególnie aktywności realizowanych za pomocą technik komputerowych w czasie rzeczywistym, takich jak wieloosobowe gry i wideokonferencje, lecz również – realizowanych za pomocą innych mediów, na przykład rozmów telefonicznych.

Drugi z czynników określających stopień interaktywności to jej zakres, rozumiany jako liczba atrybutów, które może modyfikować użytkownik. Im więcej parametrów w danym środowisku może podlegać zmianom, tym większy zakres interaktywności. Steuer według tego kryterium klasyfikuje media wizualne. Na początku skali umieszcza tradycyjnie rozumianą telewizję, która może być jedynie włączona i wyłączona, następnie nagrania na

kasetach wideo, których części mogą być wielokrotnie odtwarzane, i dalej – treści nagrane na nośnikach cyfrowych (płytach DVD, Blu-ray, dyskach twardych i tym podobnych), które można dodatkowo zwalniać lub przyspieszać, dokonywać zbliżeń, a nawet modyfikacji poszczególnych ujęć. Medium o największym stopniu interaktywności jest natomiast, jego zdaniem, interaktywna animacja komputerowa, pozwalająca użytkownikowi na interakcje z nią w czasie rzeczywistym.

Mapowanie z kolei to sposób, w jaki ludzie oddziałują na środowisko. Z jednej strony, działania te mogą być sztuczne, to znaczy zupełnie niezwiązane z ich odpowiednikami w świecie rzeczywistym, jak na przykład regulowanie głośności telewizora za pomocą ruchów palców u stopy, z drugiej – zupełnie naturalne, jak kierowanie wirtualnym samochodem za pomocą kierownicy.

Wszystkie trzy czynniki opisane przez Steuera – szybkość, zakres i mapowanie – określają łatwość wchodzenia w interakcję, obejmującą wszystkie media. Z tego względu za interaktywne należy uznać, podobnie jak film, również radio, książki, fotografie i tradycyjne formy sztuki (jak malarstwo czy rzeźba). Takie ujęcie interaktywności, choć umożliwiające łatwe porównywanie mediów, wydaje się być jednak sprzeczne z intuicją.

Inną definicję, bliższą potocznemu rozumieniu tego terminu, zaproponował Sitarski (2002), który pisze: „uważam interaktywność za pewną własność systemów złożonych z maszyn i posługujących się nimi ludzi, która zapewnia tym systemom sprzężenie zwrotne" (s. 36). Sprowadza on zatem interaktywność do wywodzącego się z cybernetyki pojęcia sprzężenia zwrotnego[9]. Zauważa przy tym, że układ komputer–użytkownik jest niemal zawsze interaktywny, co wynika z samej zasady działania komputera.

Do zaistnienia sprzężenia zwrotnego konieczne jest następstwo pewnych działań, a więc interaktywność charakteryzuje tylko te systemy, które są zorganizowane w oparciu o chronologię; można to traktować jako wadę przedstawionej definicji. Nie pozwala ona również, w odróżnieniu od definicji Steuera, na stopniowanie pojęcia interaktywności: dany system jest interaktywny – bądź też nie.

4.2.3. Sztuczność

Trzecią cechą wirtualnej rzeczywistości wymienioną przez Heima jest *sztuczność* (ang. *arificiality*). Środowisko komputerowe rozumiemy jako

[9] Pojęcia cybernetyczne wykorzystuje się również do definiowania innych zjawisk związanych z informatyką, na przykład do określenia pojęcia „informacja" (por. paragraf 3.2.3).

„sztuczne" w opozycji do naturalnej rzeczywistości. Jednakże obecnie coraz trudniej jest podać jednoznaczne kryterium dla odróżnienia pojęć: naturalne/sztuczne. Otaczająca nas rzeczywistość jest bowiem w dużej mierze wytworem działalności człowieka, a nie tylko sił natury. Dlatego też rozumienie terminu „sztuczny" w taki sposób, by obejmował on wszystko, co jest wytworem człowieka, okazuje się zbyt szerokie. „Sztuczność" powinna być raczej traktowana jako jedna z wielu własności wirtualnej rzeczywistości – z podkreśleniem, że jest ona generowana komputerowo. Gdyby zaś uznać ją za pojęcie definiujące, tym samym można by twierdzić, że wszyscy żyjemy w wirtualnej rzeczywistości i to nieustannie, a nie tylko kiedy obcujemy z komputerami.

4.2.4. Immersja

Jeśli próbowalibyśmy wskazać cechę definiującą rzeczywistość wirtualną, to na szczególną uwagę zasługuje immersja (*ang. immersion*) – rozumiana jako „zanurzenie w". Zdaniem wielu badaczy wyróżnia ona świat wirtualny spośród innych systemów symulacyjnych. Randal Walser, jeden z twórców systemów wirtualnej rzeczywistości, twierdzi, że „[d]ruk i radio mówią; scena i ekran pokazują, podczas gdy rzeczywistość wirtualna zanurza" (za: Rheingold, 1991, s. 192). Roman Konik (2009) natomiast zauważa: „[o]becnie terminem oddającym najwierniej ideę wirtualnej rzeczywistości jest immersja (zanurzenie) w alternatywnych światach stworzonych przy pomocy komputerów" (s. 93). Do badaczy niezgadzających się z powyższym poglądem należy Gurczyński. Pisze on (2013):

> Rola pojęcia „immersji" przy określaniu rzeczywistości wirtualnej wydaje się drugorzędna. Zanurzenie w danym środowisku jest możliwe dzięki specyficznym cechom środowiska – w szczególności w przypadku rzeczywistości wirtualnej kluczowe znaczenie ma możliwość podejmowania napotykających na opór działań i rozwiązywania problemów. Zjawisko immersji jest wtórne w stosunku do bardziej podstawowych – umożliwiających zanurzenie – własności danego środowiska. Jako takie pojęcie „immersji" jest również pochodne w odniesieniu do bardziej podstawowych terminów charakteryzujących środowiska wirtualne (s. 145).

Spróbujmy sprecyzować pojęcie „immersja", tak różnie rozumiane w odniesieniu do wirtualnej rzeczywistości. Ma ona dwa wymiary: technologiczny oraz psychiczny. Rozpocznijmy od technologii.

Wywołaniu wrażenia zanurzenia zmysłowego w wirtualnym świecie służą: specjalne okulary lub hełmy z wmontowanymi wyświetlaczami (ang.

HMD[10]), rękawice cyfrowe (ang. *data gloves*) oraz inny sprzęt tłumaczący ruchy ciała, oczu i rąk na dane wprowadzane do komputera. Sprzęt taki umożliwia użytkownikowi interakcję z wirtualną rzeczywistością na wiele sposobów – podobnie jak dzieje się to w rzeczywistości realnej poprzez: oglądanie, mówienie, słuchanie, dotykanie czy przemieszczanie się w jej obrębie. Warto zauważyć, że niektóre typy hełmów nie tylko wyświetlają obrazy i emitują dźwięki z wirtualnej rzeczywistości, ale jednocześnie całkowicie odcinają użytkownika od bodźców zewnętrznych, docierających z rzeczywistości. Podobnie na przykład wirtualne kokpity w samolotach wyświetlają pilotowi uproszczony, generowany komputerowo model otaczającego go świata. Można zatem stwierdzić, że technologia nie tylko umożliwia użytkownikowi zanurzenie się w świat wirtualny, ale również ogranicza, a nawet całkowicie eliminuje bodźce docierające do niego ze świata realnego. Najpopularniejsze systemy rzeczywistości wirtualnej nie oferują jednak pełnego zastąpienia bodźców pochodzących z rzeczywistości sygnałami komputerowymi, dlatego też niektórzy badacze proponują określać je jako „mieszaną rzeczywistość wirtualną". Można by nawet pokusić się o przedstawienie systemów wirtualnych na skali, której jednym krańcem jest „rzeczywistość" (realność), a drugim – „czysta wirtualność", z systemami „mieszanymi" gdzieś pośrodku.

Ze zjawiskiem immersji mamy do czynienia również w przypadku innych mediów, niezwiązanych z tak wyspecjalizowaną technologią. Oglądając filmy, czytając książkę lub słuchając muzyki, możemy siłą wyobraźni zanurzyć się w inny, fikcyjny świat. Jest to jednak nie tyle doznanie fizyczne, co psychiczne. Gurczyński (2013) pisze: „Zanurzenie mentalne jest więc odczuciem czysto psychicznym, stanem emocjonalnym, cechującym się zawieszeniem niewiary i głębokim zaangażowaniem w odbierane treści" (s. 143).

Ciekawy aspekt immersji pojawia się w związku z pewnym rodzajem gier komputerowych – nazywanych z języka angielskiego MUD-ami (ang. *Multi-User Dungeons*) lub po polsku „Wieloosobowymi Lochami" – których nadrzędnym celem jest wywołanie u użytkownika odczucia przebywania w innym świecie. Gry te nie są związane z żadnym sprzętem, co więcej – nie ma w nich nawet wyrafinowanej grafiki. Są to systemy, w których komunikacja z komputerem odbywa się za pomocą interfejsu tekstowego. Celem rozgrywki w MUD-ach jest pokonanie przeciwnika lub zdobycie największej liczby monet czy punktów. Gra polega na tym, że użytkownik przemierza jej świat, wykonując szereg zadań i rozwiązując zagadki. Ponieważ w grze może

[10] HMD (*Head-Mounted Displays*) zostały spopularyzowane przez Jarona Laniera i jego firmę VPL Research.

uczestniczyć wielu graczy jednocześnie, możliwe jest ich współdziałanie w osiągnięciu określonego celu. Gry te nie mają nic wspólnego z hełmami, rękawicami czy skafandrami, a jednak ich entuzjaści i badacze nazywają je rzeczywistością wirtualną. Ich wirtualny świat tworzy się nie na ekranie czy też w przestrzeni, ale w wyobraźni każdego z graczy, który „zanurza się" w wirtualną rzeczywistość.

Z immersją związana jest kolejna cecha wirtualnej rzeczywistości nazwana przez Heima (1993) „całkowitym zanurzeniem" (ang. *full body immersion*[11]). Heim używa tego terminu w odniesieniu do środowisk interaktywnych tworzonych przez Myrona Kruegera (por. przypis 2), w których użytkownik swobodnie się porusza i z którymi wchodzi w interakcje. Jednym z najbardziej znanych jest stworzone w 1985 roku *Videoplace*. W projekcie tym Krueger, używając technik telewizyjnych, umieszczał uczestników w dwóch pomieszczeniach, które mogły być fizycznie bardzo od siebie odległe. Obrazy z tych różnych miejsc nakładano na siebie, dzięki czemu uczestnicy mieli wrażenie, że znajdują się tuż obok, że mogą się dotykać i manipulować swoją reprezentacją na ekranie. Badacz nie wykorzystywał przy tym żadnych wspomnianych wcześniej technologii służących immersji, hełmów, skafandrów bądź innych.

4.2.5. Teleobecność

Inną cechą, którą można przypisać rzeczywistości wirtualnej, jest teleobecność (ang. *telepresence*). E-mail, wideokonferencje, edukacja na odległość, a nawet telefony – to różne rodzaje tej obecności. Wszystkie związane z nią technologie umożliwiają użytkownikom porozumiewanie się – także przy bardzo dużych odległościach. Możliwe są jednak również inne, poza rozmową, formy interakcji. Na przykład w badaniach kosmosu wykorzystuje się zdalnie sterowane roboty oraz statki bezzałogowe do eksploracji odległych miejsc w układzie słonecznym. Także wojsko używa sprzętu umożliwiającego ludziom nie tylko widzenie i słyszenie „na odległość", ale również zdalne ingerowanie w środowisko za pomocą narzędzi i instrumentów. Z tak rozumianą teleobecnością mamy do czynienia coraz częściej – i to w różnych obszarach badań.

Steuer (1992) wprowadza stopniowanie teleobecności[12], rozumiane jako poczucie przebywania w środowisku z uwzględnieniem jego plastyczności i interaktywności. Proponuje on również sposób uporządkowania

[11] Gurczyński (2012) tłumaczy ten termin jako „zanurzenie pełnocielesne".

[12] Por. paragraf 4.2.2, gdzie omówiono stopniowanie interaktywności.

mediów, umieszczając je w układzie współrzędnych, na którego osiach znajdują się te właśnie dwie cechy. Medium charakteryzującym się największym stopniem plastyczności i interaktywności jest, jego zdaniem, rzeczywistość wirtualna. Oczywiście, teleobecność może być, podobnie jak inne wspomniane wcześniej cechy wirtualnej rzeczywistości, realizowana analogowo – na przykład poprzez rozmowy telefoniczne, systemy konferencyjne czy też sterowanie urządzeniami za pomocą fal radiowych. Nie jest ona zatem cechą definiującą wirtualną rzeczywistość.

4.2.6. Komunikacja sieciowa

Zdaniem Heima, ważną rolę w charakterystyce wirtualnej rzeczywistości odgrywa komunikacja sieciowa (ang. *networked communications*). Możliwość połączenia komputerów w sieć powoduje, że wielu ludzi może jednocześnie doświadczać tej samej rzeczywistości wirtualnej. To właśnie intersubiektywność, możliwość dzielenia jej z innymi użytkownikami, odróżnia rzeczywistość wirtualną od fantazji. Fantazje, marzenia i sny są bowiem doświadczeniami prywatnymi. Intersubiektywność, możliwa między innymi dzięki komunikacji sieciowej, otwiera nowe możliwości dla ludzkiej aktywności. Komunikacja, sztuka, polityka, a nawet akty seksualne i przemocy są tymi działaniami człowieka, które znalazły już swoje miejsce w tejże rzeczywistości.

Podane przez Heima cechy symulacji generowanych komputerowo – które są: interaktywne, mogą być dzielone przez wielu użytkowników, dają wrażenie całkowitego zmysłowego „zanurzenia się" w nią i które umożliwiają użytkownikom komunikację, działanie i współdziałanie na odległość – są najczęściej przytaczanymi cechami wirtualnej rzeczywistości. Oczywiście, ich listę można rozszerzać bądź też ograniczać.

4.2.7. Fikcjonalizacja

Sitarski (2002) twierdzi, że *differentia specifica* rzeczywistości wirtualnej to: zanurzenie (immersja), teleobecność oraz interaktywność. Podkreśla on przy tym szczególną rolę fikcjonalizacji:

> Rzeczywistość wirtualna daje uczestnikowi poczucie przebywania w innym środowisku. To wrażenie zanurzenia wynika przede wszystkim z interaktywności komunikacji, czyli z możliwości wprowadzania zmian w środowisku zmediatyzowanym. Inne czynniki, wśród nich przede wszystkim złudzenia zmysłowe, mają znaczenie drugorzędne. Ważne jest natomiast, by działania człowieka

i komputera były sfikcjonalizowane, to znaczy aby oprócz swojego technicznego znaczenia miały jeszcze inne, pozwalające budować fikcyjną rzeczywistość. Fikcjonalizacja umożliwia przejście od interakcji z komputerem do działania w odrębnym środowisku. Połączenie tych elementów stanowi właśnie *differentia specifica* rzeczywistości wirtualnej (s. 40).

4.2.8. Hipertekst

Gurczyński (2013) poszerza listę pojęć związanych z wirtualną rzeczywistością podaną przez Heima o hipertekst (hipertekstowość) oraz czas. Dodanie hipertekstu związane jest z twierdzeniem, że Internet jest obecnie najbardziej rozpowszechnioną rzeczywistością wirtualną, a cechą umożliwiającą jego interaktywność jest właśnie hipertekstowość. Gurczyński pisze (2013): „Hipertekst w Internecie jest dla nas czymś tak naturalnym, że praktycznie go już nie zauważamy – po prostu klikamy w kolejne odnośniki, przechodząc od informacji do informacji" (s. 155). Czym jest zatem ów „niezauważalny" hipertekst?

Hipertekst jako tekst z systemem odnośników można rozumieć jako swego rodzaju metatekst (tekst złożony z tekstów dostępnych po naciśnięciu odpowiednich odnośników)[13]. Można też podjąć próbę jego określenia przez podanie listy elementów, z których się składa. Taką definicję zaproponował Luciano Floridi (1999). Twierdzi on, że hipertekst jest konstytuowany przez trzy elementy: węzły, zestaw połączeń oraz interaktywny i dynamiczny interfejs. Węzły mogą być: dokumentami alfanumerycznymi, dokumentami multimedialnymi lub też jednostkami funkcjonalnymi (na przykład apletami). Są one połączone odnośnikami, osadzonymi w węzłach za pomocą specjalnych znaczników (nazywanych kotwicami), które umożliwiają użytkownikowi przełączanie się pomiędzy węzłami. Użytkownik operuje tymi odnośnikami za pomocą interfejsu.

Floridi twierdzi, że powyższa definicja hipertekstu pozwala uniknąć trzech najczęściej popełnianych błędów w jego określaniu. Pierwszym jest błąd literackości, polegający na traktowaniu hipertekstu jako techniki narracyjnej, podczas gdy hipertekst jest jedynie sposobem organizowania informacji. Drugi – to błąd ekspresyjności, to znaczy traktowania hipertekstu jako narzędzia ułatwiającego tworzenie tekstów – choć został on zaprojektowany dla wygody czytelnika i ma być przede wszystkim narzędziem ułatwiającym czytanie i korzystanie z tekstów. Kolejny błąd – elektroniczności – polega na stwierdzeniu, że hipertekst jest związany wyłącznie z komputerami. Oczywiście większość jego implementacji dokonuje się w środowisku

[13] Por. np. (Heim, 1993).

elektronicznym (komputery, e-booki, tablety, telefony etc.), ale możliwe są również inne jego realizacje. Na przykład autor idei hipertekstu, Vannevar Bush, opublikował w 1945 roku teoretyczny model komputera analogowego, który miał pracować na mikrofilmach, tworzyć odnośniki między nimi oraz automatycznie pomiędzy nimi przechodzić. Hipertekst jest zatem określony jako sposób organizacji tekstu, a metoda jego realizacji jest kwestią drugorzędną.

Obecnie nie ma powszechnej zgody co do sposobu rozumienia hipertekstu. Niektórzy filozofowie nie uznaliby trzech jego ujęć, przytoczonych przez Floridiego, za błędne. Często podkreśla się bowiem literacki wymiar hipertekstu i jego narracyjność, analizuje się także rolę czytelnika i pisarza.

4.2.9. Czas

Ciekawym aspektem wirtualnej rzeczywistości jest czas. Wraz z możliwością tworzenia symulacji komputerowych nastąpiła możliwość bifurkacji czasu tak, że konieczne stało się odróżnienie czasu w rzeczywistości wirtualnej od czasu poza nią (czasu rzeczywistego). Czas wirtualny może płynąć tak samo jak rzeczywisty (na przykład podczas wideokonferencji) lub też zupełnie inaczej. Można go bowiem zatrzymać, spowolnić, przyspieszyć, a nawet cofnąć. Na przykład symulację lub grę można wstrzymać lub przywrócić do stanu wcześniejszego, by umożliwić użytkownikowi ponowne wykonanie pewnych czynności. Takie zabiegi zwiększają w znacznym stopniu atrakcyjność gier symulacyjnych, takich jak The Sims[14]. Czas może się różnić także ze względu na wykorzystywaną w danej symulacji lub grze technologię, na przykład gdy do sieci podłączone są komputery o różnych parametrach technicznych (z procesorami różnej prędkości). Pewne obiekty mogą wówczas pojawiać się, zmieniać czy zanikać wcześniej na monitorach komputerów szybszych, a później – na monitorach komputerów wolniejszych.

Czas jest zatem, podobnie jak przestrzeń, cechą różniącą wirtualność od rzeczywistości. Gurczyński (2013) tak podsumowuje cechy wirtualnej rzeczywistości:

---

[14] Seria gier symulacyjnych, w których gracz ma za zadanie organizować czas kierowanym przez niego wirtualnym postaciom, zwanym Simami, i pomagać im w osiąganiu ich własnych celów rozwojowych. Jedną z głównych atrakcji w grze jest możliwość zatrzymywania czasu w dowolnym momencie oraz przyspieszanie go, aby szybciej osiągnąć pewne cele. Czas można również cofnąć, co pozwala na anulowanie wybranych (zazwyczaj przykrych) zdarzeń, do których już doszło.

> Rzeczywistość w swojej najogólniejszej formie jest określona przez przestrzeń i czas. Sfera wirtualna, stwarzając własną przestrzeń i czas, staje się nową formą rzeczywistości. Jest to rzeczywistość swoiście ludzka, gdyż ufundowana na obiektywnych procesach obliczeniowych, jej zawartość jest treścią naszych wyobrażeń (s. 159).

Przedstawione powyżej próby określenia rzeczywistości wirtualnej doprowadziły do powstania długiej listy cech ją definiujących. Znajduje się na niej wiele pojęć i zjawisk, z opisem których muszą zmierzyć się filozofowie analizujący problemy ontologiczne, epistemologiczne i etyczne związane ze światem wirtualnym. W dalszej części skoncentrujemy się przede wszystkim na dwóch kwestiach: statusie rzeczywistości (obiektów) wirtualnych oraz na sposobach odróżniania tego, co wirtualne, od tego, co rzeczywiste[15].

4.3. Obiekty wirtualne jako symulacje

Jednym z podstawowych zagadnień ontologicznych jest problem istnienia obiektów. W jaki sposób istnieją obiekty rzeczywistości wirtualnej? Czy ich sposób istnienia jest inny, czy też, w jakimś sensie, podobny do sposobu istnienia obiektów otaczającej nas rzeczywistości? To tylko niektóre pytania, na jakie próbują odpowiedzieć badacze rzeczywistości wirtualnej.

Obiekty wirtualne są bardzo często traktowane jako przybliżenia bądź też symulacje obiektów rzeczywistych. Przez „symulacje" rozumie się naśladowanie, imitowanie lub odwzorowanie jakiegoś oryginału. Takie ich ujęcie jest, w pewnym stopniu, związane bezpośrednio z językiem, jakim się posługujemy. Znaczenie słowa „wirtualny", w kontekście wirtualnej rzeczywistości, jest pochodnym znaczenia angielskiego słowa *virtual*. Przymiotnik ten oznacza „faktycznie taki, prawie pod każdym względem" (*being actually such in almost every respect*), na przykład – budynek jest praktycznie ruiną (*the building is a virtual ruin*). Zatem używanie go w wyrażeniu „obiekt wirtualny" sugeruje, że wirtualne *x* jest w pewnym sensie przybliżeniem (symulacją) lub jest prawie rzeczywistym (prawdziwym) *y*. Pojawia się jednak pytanie, co rozumieć przez „rzeczywiste (prawdziwe) *y*". Wydaje się, że takie traktowanie obiektów wirtualnych niesie ze sobą więcej pytań niż odpowiedzi.

Po pierwsze, określenie jakiegoś obiektu jako rzeczywistego (prawdziwego) nie zawsze jest jednoznaczne. Coś, co jest nierzeczywiste (niepraw-

---

[15] Prezentowany tu przegląd nie wyczerpuje wszystkich poglądów ontologicznych dotyczących omawianych kwestii. Przedstawiamy te z nich, które w literaturze pojawiają się najczęściej.

dziwe) w jednym opisie, może być prawdziwe w innym, na przykład gumowa kaczka zabawka – jest prawdziwą zabawką, ale nie jest prawdziwą kaczką.

Po drugie, jeśli przez symulację rozumie się naśladowanie, imitowanie lub odwzorowanie jakiegoś oryginału, to problematyczne staje się określenie przypadku, gdy takiego oryginału nie ma, czyli gdy wirtualne obiekty lub zdarzenia nie mają swoich odpowiedników rzeczywistych. Jak zauważają Crandall i Levich (1998), w doświadczeniach wirtualnych mamy czasem do czynienia z obiektami i sytuacjami rzeczywistymi, a czasem – z fikcyjnymi. Przebywanie w symulatorze lotów jest przykładem doświadczeń pierwszego rodzaju. Natomiast obcowanie z postaciami fikcyjnymi, jak na przykład głaskanie potwora w wirtualnej rzeczywistości, nie ma nic wspólnego z doświadczaniem rzeczywistości. Sugestia, że obiekty i doświadczenia wirtualne są przybliżeniami ich odpowiedników w świecie rzeczywistym, nie ma zastosowania do wszystkich przypadków, na przykład do tych, w których nie istnieje odpowiadające im doświadczenie lub obiekt rzeczywisty. W pozostałych przypadkach, to znaczy wtedy, gdy wirtualna rzeczywistość jest zbiorem obiektów i doświadczeń, jakich może doświadczać człowiek w realnym świecie, mówimy o tak zwanej ugruntowanej rzeczywistości wirtualnej (ang. *grounded virtual reality*). Jej przykładem jest wspomniany już symulator lotów, ponieważ naśladuje on (przybliża) lot samolotem, który można odbyć w rzeczywistym świecie.

Po trzecie, jak twierdzą Crandall i Levich, to, co próbuje się symulować w wirtualnej rzeczywistości, to nie tylko same obiekty, ale również pewne warunki zachodzące pomiędzy postrzeganym obiektem a podmiotem poznającym. W warunkach tych podmiot będzie doświadczał obiektu wirtualnego w sposób podobny do tego, w jaki doświadcza on rzeczywistości.

Wraz z przyjęciem tezy, że obiekty wirtualne są symulacjami obiektów rzeczywistych, pojawiają się nowe pytania, na które należy znaleźć odpowiedź. W jaki sposób i gdzie obiekty te powstają? Jaki status ontologiczny mają owe symulacje? Czy są one tak samo rzeczywiste jak obiekty otaczającego nas świata?

Na pierwsze z tych pytań można odpowiedzieć na różne sposoby. Z materialistycznego punktu widzenia, rzeczywistość wirtualna to tylko wytwór obwodów i kabli. Można jednak rozumieć ją szerzej, nie ograniczając się tylko do sfery technologii: „na świat wirtualny składają się nie tylko realizacje informatyczne […], ale też świat sztuki, filmu, muzyki, gier komputerowych, projektów badawczych, myśli, wyobrażeń realizowanych na drodze wizualizacji" (Latawiec, 2009, s. 53). Tak rozumiany świat wirtualny jest dziełem człowieka, ale źródłem jego jest nie tylko technologia, lecz także

wiedza, doświadczenia, wyobraźnia, marzenia, a nawet sny[16] jego twórcy. Pierwotnym źródłem świata wirtualnego jest rzeczywistość empiryczna lub świat obiektów abstrakcyjnych (pojęcia ogólne, idee Platona i temu podobne), a wtórnym – świat ludzkich myśli. Anna Latawiec (2009) pisze:

> Przez świat wirtualny rozumiem obraz rzeczywistości kreowany lub odkrywany przez człowieka na drodze symulacji intelektualnej lub technicznej. Tak pojęty świat jest tworem człowieka. Swoje źródło ma on w szeroko pojętej rzeczywistości. Przez rzeczywistość rozumiem zarówno tę empiryczną, która jest dostępna jedynie w ograniczonym zakresie obserwatorowi, jak i „rzeczywistość samą w sobie", czyli istniejącą poza zasięgiem obserwatora (s. 52).

Skoro świat wirtualny jest dziełem człowieka i wiemy, jakie są źródła jego kreacji, trzeba postawić pytanie, gdzie powstają obiekty tego świata. Elisabeth Reid (1994) odpowiada na nie następująco: „Cyberprzestrzeń – kraina impulsów elektronicznych i ekspresowych autostrad danych, gdzie istnieją MUD-y – jest być może technicznym artefaktem, ale rzeczywistość wirtualna stanowi konstrukcję znajdującą się wewnątrz ludzkiego umysłu. […] Iluzja rzeczywistości nie spoczywa w samej aparaturze, ale w chęci użytkowników, by wytwory ich wyobraźni traktować tak, jakby były rzeczywiste" (s. 6-7). Na ową intencjonalność istnienia wirtualności zwraca uwagę również Tadeusz Miczka (2009), pisząc: „VR [wirtualna rzeczywistość] jest przecież sztucznym tworem, który istnieje intencjonalnie, dlatego że człowiek chce obdarzyć go istnieniem" (s. 19). Warto zatem rozważyć związki pomiędzy obiektami wirtualnymi a jednym z rodzajów obiektów intencjonalnych – mianowicie obiektami fikcyjnymi.

4.4. Wirtualność a fikcja

W wielu analizach pojęcia „wirtualności" napotkać można jego zestawienie z pojęciem „fikcji". Może to być spowodowane faktem, że w wirtualnym świecie doświadczamy rzeczy, jak choćby wspomniane już głaskanie potwora, które nie mają swojego odpowiednika w rzeczywistości – są to doświadczenia z obiektami fikcyjnymi. Rozważania tego typu związane są najczęściej z analizą światów gier komputerowych.

Mianem „obiektów fikcyjnych" określa się zazwyczaj postaci, przedmioty lub zdarzenia, które pojawiają się w dziełach literackich lub filmowych,

---

[16] Często twierdzi się, że doskonałym przykładem świata wirtualnego, poza światami generowanymi komputerowo, są sny. Szczególnie że sen może być (i często jest) zafałszowaniem świata realnego.

jak choćby Sherlock Holmes, latający dywan czy Kaczor Donald. Są one przykładami przedmiotów nieistniejących lub, mówiąc językiem Ingardena, „przedmiotów przedstawionych". Oczywiście nie muszą to być tylko obiekty znane z literatury lub filmu, ale również wszystkie inne wytwory ludzkiej świadomości funkcjonujące w świecie, takie jak na przykład Święty Mikołaj czy też kwadratowe koło.

Obiekty rzeczywistości wirtualnej, a w szczególności postaci z gier komputerowych, wydają się pod wieloma względami przypominać obiekty fikcyjne. Przyjrzyjmy się zatem cechom tych ostatnich. Jeśli bowiem elementy świata wirtualnego można traktować jak fikcje, to do stworzenia ich ontologii warto wykorzystać istniejące teorie obiektów fikcyjnych, jak choćby teorię obiektów Meinonga (1960) czy też teorię przedmiotów czysto intencjonalnych Ingardena (1960-1961, 1988).

Gurczyński (2013) pisze:

> Przedmioty fikcyjne są przedmiotami z konieczności nieistniejącymi (realnie), lecz twierdzenie to należy odróżnić od stwierdzenia, że przedmioty fikcyjne nie mają żadnego statusu ontologicznego. Podstawowym założeniem, pozwalającym na przyznanie obiektom fikcyjnym pewnego statusu egzystencjalnego […] jest teza Franza Brentana mówiąca, że każdy akt świadomości ma swój przedmiot, czyli, inaczej mówiąc, każdy akt świadomości jest **intencjonalny**. Zawsze, gdy mówimy, mówimy o **czymś**, nasze fantazje dotyczą **czegoś**, patrzymy na **coś**, myślimy o **czymś** – wszelkie przeżycie świadome jest zawsze „świadomością czegoś" (s. 183).

Przedmioty fikcyjne są zatem intencjonalne. Zdaniem Ingardena są obiektami „czysto intencjonalnymi", które istnieją w specyficzny – intencjonalny – sposób. Są one zależne w swym istnieniu od wytwarzających je aktów świadomości, a więc od innych przedmiotów – są bytowo heteronomiczne, co odróżnia je od obiektów rzeczywistych, które istnieją autonomicznie. Co więcej, przedmiot fikcyjny zyskuje swój ontologiczny status w czasie późniejszym niż tworzący go akt świadomości. Owe następstwo czasowe bytów jest, zdaniem Ingardena, cechą, która nie pozwala traktować obiektów fikcyjnych jako przedmiotów idealnych lub abstrakcyjnych, czyli pozaczasowych.

Obiekty fikcyjne są szczególnym rodzajem przedmiotów intencjonalnych – są bowiem intersubiektywne. Nie wszystkie wytwory ludzkiej świadomości, na przykład sny, możemy przecież dzielić z innymi. Intersubiektywność przedmiotów czysto intencjonalnych zapewnia podstawa bytowa, od której przedmioty te są zależne. W przypadku utworu muzycznego jest to partytura, dla utworu literackiego – jego tekst, rozumiany jako ciąg symboli języka.

Warto zauważyć, że podstawa bytowa może być zarówno analogowa (klasyczna książka), jak i cyfrowa (e-book).

Według Ingardena (1988) przedmioty fikcyjne powstają na skutek prostego przedstawienia – „w swej całości". Z kolei Meinong (1960) uważa, że stoją one poza istnieniem czy nieistnieniem, że są to przedmioty obiektywnie możliwe[17]. Jednak obaj autorzy zgadzają się co do dwóch podstawowych cech obiektów fikcyjnych: są one określone w sposób niezupełny oraz mogą posiadać sprzeczne cechy.

Niezupełność określenia[18] (niezupełność ontologiczna) – rozumiana jako stwierdzenie, że dla pewnych cech przedmiotu nie możemy stwierdzić, czy posiada on tę cechę, czy też nie – jest cechą charakterystyczną przedmiotów fikcyjnych, odróżniającą je od obiektów realnych. Nie umiemy na przykład określić koloru oczu bohatera powieści, o ile nie został on podany przez jej autora. Wynika to z faktu, że w danym utworze podać można jedynie skończoną liczbę cech każdego przedmiotu. Oczywiście, nie mamy również pełnej wiedzy dotyczącej niektórych postaci lub faktów rzeczywistych – na przykład historycznych, jest to jednak niekompletność epistemologiczna, a nie ontologiczna. Jan III Sobieski miał zielone oczy lub też nie, ale możemy tego nie wiedzieć ze względu, na przykład, na niekompletność źródeł historycznych.

Wśród przedmiotów fikcyjnych wyróżnić można te, które posiadają sprzeczne własności, jak na przykład kwadratowe koło czy bezdzietna matka. Nazywa się je przedmiotami sprzecznymi, przy czym słowo „sprzeczność" należy tu rozumieć w sposób konieczny[19].

Wszystkie obiekty fikcyjne natomiast mają jeszcze jedną, bardzo ważną własność – charakteryzują się dwupodmiotowością – mówiąc inaczej, dwupoziomowością określenia[20]. I tak o Pinokiu możemy powiedzieć, że jest on drewnianym pajacem, że wydłuża mu się nos, gdy kłamie. Posiada on bowiem te własności w świecie fikcyjnym stworzonym przez Carla Collodiego. Takie własności obiektu nazywa się „wewnętrznymi". Z kolei jego cechami „zewnętrznymi" są: bycie postacią fikcyjną, bycie tworem wymyślonym przez Carla Collodiego, jak też niezupełność ontologiczna. Zazwyczaj uważa się, że cechy zewnętrzne przedmiotów świadomości są określone w sposób zupełny, podobnie jak przedmiotów realnych. Co ważne, „pomijając ową dwupoziomowość uposażenia przedmiotów fikcyjnych, zmuszeni bylibyśmy

---

[17] Analizę teorii Meinonga i Ingardena dla obiektów fikcyjnych znaleźć można np. w pracach *Alexius Meinong i Roman Ingarden o intencjonalności i przedmiotach fikcyjnych*; *Czym jest wirtualność...* (Gurczyński, 2004, 2013).

[18] Gurczyński (2013) określa te cechę mianem „niezupełności uposażenia".

[19] Niektórzy filozofowie negują sensowność badań nad takim obiektami, na przykład poprzez wyeliminowanie ich z dziedziny przedmiotów intencjonalnych.

[20] Por. Gurczyński (2013), s. 193-196.

uznać, że każdy przedmiot fikcyjny jest sprzeczny, będąc zarazem fikcyjnym (w świecie realnym) i realnym (w świecie fikcyjnym)" (Gurczyński, 2013, s. 194).

Przyjrzyjmy się teraz tym z wymienionych cech, które przysługują obiektom wirtualnym, co pozwoli nam stwierdzić, czy dziedzina tych obiektów pokrywa się z dziedziną obiektów fikcyjnych.

Obiekty wirtualne, podobnie zresztą jak fikcyjne i realne, są intersubiektywnie dostępne – nie są prywatnymi doznaniami podmiotu, jak na przykład sny. Postrzegane z zewnątrz, charakteryzują się również dwupoziomowością określenia. Rozważmy dla przykładu postaci z gier komputerowych. Mają one dwa rodzaje cech: wewnętrzne, które posiadają jako elementy świata wirtualnego (świata gry) – wygląd, cechy charakteru, umiejętności, płeć i tym podobne, oraz zewnętrzne, przysługujące im z perspektywy rzeczywistości (spoza gry) – bycie obiektem gry, bycie wirtualnym, bycie stworzonym przez twórców tejże gry. Gurczyński (2013) sugeruje, że ową dwupoziomowość określenia można uogólnić: „zawsze wówczas, gdy świat s_2 jest nadbudowany nad (jest ontycznie zależny) światem s_1, to przedmioty ze świata s_2, postrzegane z poziomu świata s_1, będą charakteryzowały się dwupoziomowością uposażenia" (s. 199). Wtedy cecha ta jest wspólną własnością obiektów fikcyjnych i realnych.

Inną wspólną cechą przedmiotów fikcyjnych i obiektów rzeczywistości wirtualnej jest ich niezupełność ontologiczna. Na przykład dla wspomnianych już postaci z gier komputerowych nie sposób określić tych własności, których nie określili ich twórcy. Co więcej, zaprojektowanie takiego środowiska wirtualnego, w którym choć kilka obiektów miałoby cechy określone w sposób zupełny, wydaje się niemożliwe. Rzeczywistość wirtualna jest bowiem realizowana przez procesy obliczeniowe, które nie mogą zawierać nieskończoności aktualnych, a zatem również nieskończonej liczby cech jej obiektów.

Obiekty wirtualne są zaprojektowane intencjonalnie i realizowane przez procesy obliczeniowe zachodzące w komputerach, które są transcendentne w stosunku do kreowanych środowisk wirtualnych. Innymi słowy, komputery nie są elementem rzeczywistości wirtualnej, choć są niezbędne do przeprowadzania procesów obliczeniowych. Widać tu pewną analogię z obiektami fikcyjnymi (intencjonalnymi), zależnymi ontologicznie od aktów świadomości, które to akty są transcendentne w stosunku do tworzonych przez nie rzeczywistości. Zatem fikcje zależne są od aktów świadomości, a te z kolei od podmiotów – podobnie jak istnienie obiektów wirtualnych zależy od procesów obliczeniowych, a te – od komputerów.

Istnieją jednak takie cechy obiektów wirtualnych, które w zasadniczy sposób odróżniają je od fikcyjnych. Gurczyński twierdzi, że jedną z nich jest sprzeczność: obiekty wirtualne nie mogą być określone przez cechy

sprzeczne, ponieważ nie można zrealizować wizualizacji takich przedmiotów. Wydaje się jednak, że jest to wniosek fałszywy. Jeśli bowiem przyjąć, że przykładem wirtualnej rzeczywistości są wspomniane wcześniej MUD-y, to obiekty występujące w tych grach (obiekty wirtualne), które nie są wizualizowane, lecz charakteryzowane przez słowny opis, mogą być sprzeczne, gdyż taki właśnie może być ich opis.

Cechą, która bez wątpienia odróżnia obiekty wirtualne od fikcyjnych, jest dynamiczność. Istnienie świata fikcyjnego jest statyczne; światy te, raz przedstawione czy opisane przez podmiot, zmieniają się tylko w wyniku ingerencji ich twórcy, na przykład w kolejnych tomach powieści możemy poznawać nowe aspekty świata w niej przedstawionego. Żaden przedmiot fikcyjny nie jest zdolny zmieniać się „sam z siebie". Świat wirtualny natomiast jest dynamiczny, jego obiekty zaprojektowane są intencjonalnie (świadomościowo) – ale realizowane przez procesy obliczeniowe zachodzące w komputerach. Jako takie, mogą zmieniać się bez udziału aktów świadomości ich twórców, na przykład dzięki zastosowanym podczas ich implementacji algorytmom genetycznym.

Interaktywność, będąca kluczową cechą wirtualnej rzeczywistości, jest kolejną jej własnością, która nie przysługuje światom fikcyjnym. Obiekty fikcyjne są nam dane jedynie poprzez swoje cechy, nie możemy wchodzić z nimi w relacje podobne jak z przedmiotami. Natomiast obiekty wirtualne nie tylko poddają się działaniom użytkowników, ale również reagują na nie, co daje wrażenie interakcji podobnych do tych, jakie zachodzą w rzeczywistości (w realnym świecie). Im bardziej realistyczne są działania na obiektach wirtualnych, tym lepszy (bardziej realistyczny) jest świat, w którym się one znajdują (por. paragraf 4.2.2).

Podsumowując, można stwierdzić, że obiekty wirtualne i fikcyjne mają wiele cech wspólnych, takich jak: intencjonalność i związana z nią heteronomiczność ontologiczna, uzyskiwanie statusu ontologicznego później niż stwarzające je procesy, dwupoziomowość określenia i niezupełność ontologiczna. Istnieją jednak zasadnicze cechy obiektów wirtualnych, które nie przysługują obiektom fikcyjnym. Są to: interaktywność oraz ufundowanie na procesach obliczeniowych (a więc cyfrowych). Między innymi na tych cechach oparł swoją ontologię obiektów wirtualnych Gurczyński.

4.5. Ontologia rzeczywistości wirtualnej Gurczyńskiego

Jacek Gurczyński w książce *Czym jest wirtualność. Matrix jako model rzeczywistości wirtualnej* (2013) zaproponował ujęcie środowisk wirtualnych jako rzeczywistości, której status definiują dwie cechy: 1) rzeczywistość ta

umożliwia wchodzenie w ucieleśnione interakcje, przy czym „poczucie rzeczywistości wytwarza się dzięki możliwości podejmowania działań napotykających opór" (s. 233), oraz 2) opór ten jest wynikiem obiektywnych praw fizycznych zachodzących w danym środowisku. Przyjrzyjmy się bliżej tej koncepcji.

Rozpocznijmy od ważnego rozróżnienia, jakie przyjmuje Gurczyński. Pojęcie „realności" wiąże on ze światem materialnym, zewnętrznym. Z kolei „rzeczywistość" rozumie jako dowolną przestrzeń (fizyczną), w której mogą zachodzić ucieleśnione działania, służące rozwiązywaniu problemów, podczas podejmowania których podmiot napotyka na opór. Dzięki temu określenie „wirtualna rzeczywistość" nie jest oksymoronem. Autor twierdzi przy tym, że możliwość podejmowania w świecie wirtualnym działań podobnych do tych, które zachodzą w świecie realnym (materialnym), jest wystarczającym powodem, by nazywać go „rzeczywistością".

Gurczyński przyjmuje „emergentystyczny obraz rzeczywistości", w którym wszystkie struktury i procesy złożone ufundowane są na prostszych układach, czyli są w stosunku do nich ontologicznie pochodne. Środowiska wirtualne są ufundowane na procesach obliczeniowych, a o ich specyfice decyduje cyfrowość. Zatem ważną rolę w jego ontologii odgrywa aspekt technologiczny. Badacz stwierdza ponadto, że „[n]ie wiemy, czy fundamentalne procesy konstytuujące świat realny mają naturę obliczeniową/cyfrową, czy analogową – rozstrzygnięcie tego problemu ma zasadnicze znaczenie zarówno dla ontologii świata realnego, jak i dla rzeczywistości wirtualnej" (s. 209). Jeśli bowiem świat realny ma charakter ciągły (jest konstytuowany przez analogowe procesy kwantowo-mechaniczne), to zawiera on aktualne nieskończoności, na przykład czas i przestrzeń są ciągłe i nieskończenie podzielne; jeśli natomiast jego podstawę stanowią procesy obliczeniowe, to ma on naturę dyskretną, w szczególności – wszystkie realne byty są skończenie złożone.

Gurczyński (2013) pisze: „przedmioty wirtualne mają swoją genezę w świadomości podmiotu – są projektowane świadomościowo i opisywane w jakimś (sztucznym) języku programowania – a realizowane przez procesy obliczeniowe zaimplementowane w komputerach, które również są wytworem człowieka" (s. 212). Tak rozumiane obiekty wirtualne są lepiej określone niż realne, ponieważ wiemy o nich z całą pewnością, że są ufundowane przez procesy obliczeniowe, a zatem są to obiekty cyfrowe. Co więcej, podczas gdy podmiot wytwarzając realny przedmiot według swojego projektu, przekształca jedynie zastaną rzeczywistość (operuje bowiem dostępnymi materiałami, z których go buduje), to w przypadku obiektu wirtualnego zarówno projekt, jak i sposób jego realizacji mają swoje źródło w podmiocie (procesy obliczeniowe są również wytworem człowieka).

Znana jest zatem natura procesów konstytuujących obiekty wirtualne, podczas gdy natura obiektów realnych pozostaje zagadką.

Dla potwierdzenia tezy o lepszym określeniu obiektów wirtualnych (i fikcyjnych) niż przedmiotów realnych Gurczyński analizuje zasadę nazywaną prawem nieodróżnialności Leibniza, omówioną w paragrafie 1.1.2, według której do stwierdzenia odmienności dwóch obiektów wystarczy wskazanie takich cech, które posiada jeden z nich, a których nie posiada drugi. W przypadku makroskopowych przedmiotów realnych identyfikacja ich poprzez własności nie przysparza problemów. Przyjmując twierdzenie, że przedmioty realne posiadają nieskończenie wiele własności – lub tak wiele, że nie ma możliwości uwzględnienia ich wszystkich, niemożliwe jest sformułowanie ogólnego kryterium identyfikacji takich przedmiotów. W przypadku przedmiotów fikcyjnych nie ma takiej trudności, gdyż są one niezupełne ontologicznie, a zatem zawsze posiadają skończoną liczbę cech. Istnieje wówczas (teoretyczna) możliwość sformułowania takiego kryterium.

Kwestia identyczności obiektów wirtualnych jest dość złożona, ze względu na dwupoziomowość ich określania (por. podrozdział 4.4). W przypadku, gdy dokonujemy identyfikacji z poziomu uczestnika wirtualnej rzeczywistości (na przykład z poziomu gry), następuje ona poprzez podanie cech tych obiektów, dokładnie tak samo jak w przypadku obiektów realnych. Jeśli jednak identyfikuje się obiekty z poziomu obserwatora lub projektanta danej symulacji, to istnieją co najmniej dwa kryteria ich identyczności: podanie własności lub też ich nierozróżnialność na fundamentalnym poziomie obliczeniowym. Drugie z kryteriów zmuszeni jesteśmy zastosować na przykład w odniesieniu do kopii obiektów wirtualnych. Zmultiplikowane obiekty uznajemy za identyczne, gdy są zakodowane za pomocą takich samych ciągów bitów. Zatem jeśli świat realny uznajemy za analogowy, to kryteria identyczności obiektów tego świata i obiektów wirtualnych różnią się. Z kolei jeśli świat jest w swej naturze cyfrowy, to określenie kryterium identyczności jego obiektów związane jest z koniecznością przyjęcia perspektywy obserwatora, co wydaje się niemożliwe, ponieważ jesteśmy częścią tego świata. Można zatem stwierdzić, że przy obecnym stanie wiedzy (braku rozstrzygnięcia, czy świat realny jest analogowy czy cyfrowy) kryteria identyczności dla obiektów wirtualnych są lepiej określone niż dla przedmiotów realnych.

Przedmioty wirtualne mają jeszcze jedną przewagę nad realnymi. Gurczyński (2013) twierdzi, że „strefa wirtualna tworzy odrębną – lecz rzeczywistą – wobec realności sferę ontologiczną. W jej obrębie możemy wchodzić w interakcje z przedmiotami wirtualnymi na podobnej zasadzie, jak z przedmiotami ze świata realnego" (s. 219). Dodając do tego twierdzenia tezę, że u podstaw świata wirtualnego leżą procesy cyfrowe, można stwier-

dzić, że dostęp do obiektów wirtualnych (w tym operowanie na nich) jest dużo łatwiejszy niż do realnych, ze względu na ciągłą naturę tych drugich. Widać to wyraźnie na przykładzie duplikowania przedmiotów. W przypadku makroskopowych obiektów realnych stworzenie ich wiernych kopii jest właściwie niemożliwe. Duplikowanie obiektów cyfrowych jest natomiast bardzo proste – nietrudno utworzyć dwa identyczne obiekty różniące się jedynie czasem powstania oraz miejscem występowania. Pojawia się jednak pytanie, w jaki sposób odróżnić obiekty zduplikowane w wirtualnej rzeczywistości. Jest to możliwe dzięki własnościom zewnętrznym, nazywanym przez Gurczyńskiego topologicznymi, takim jak czas utworzenia czy miejsce zapisania na nośniku danych; wszystkie inne ich cechy są identyczne.

Powodem łatwiejszego dostępu do obiektów wirtualnych niż do realnych jest fakt, że strefa wirtualna jest pochodna ontologicznie w stosunku do realności. Dlatego też z poziomu świata realnego rzeczywistość wirtualną można dowolnie kształtować, a w tym – ustalać jej warunki fizyczne: przebieg czasu oraz obowiązujące w niej prawa. Gdy zostaną one już ustalone, obowiązują w niej tak samo jak prawidłowości i ograniczenia w świecie realnym. Zatem pod względem funkcjonalnym świat wirtualny jest pewnym rodzajem rzeczywistej przestrzeni.

Przestrzeń wirtualną można traktować jako rzeczywistą w sensie proponowanym przez Gurczyńskiego, to znaczy jako przestrzeń (z ograniczeniami fizycznymi), w której mogą zachodzić ucieleśnione działania napotykające na opór, służące rozwiązywaniu problemów. Takie funkcyjne rozumienie rzeczywistości prowadzi do wniosku, że „wirtualna przestrzeń zdaje się przestrzenią **rzeczywistą** i to przestrzenią funkcjonalnie nawet przewyższającą sferę świata realnego. Aspekt funkcjonalny rzeczywistości wirtualnej na obecnym etapie dociekań jest, być może, nawet ważniejszy od kwestii formalnych i substancjalnych" (s. 221).

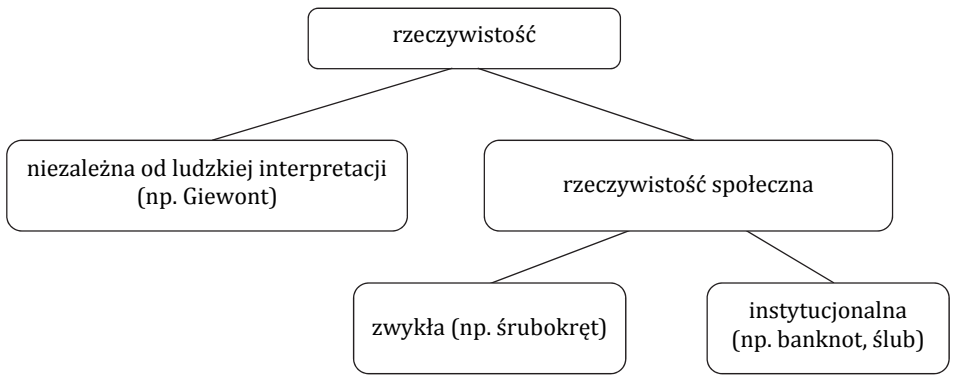
Przy takim rozumieniu wirtualności powstaje pytanie o związki zachodzące pomiędzy sferą realną a wirtualną. Rzeczywistość wirtualna jest zależna od realności, przy czym można rozpatrywać ową zależność w trzech aspektach. Po pierwsze, obiekty wirtualne realizowane są z wykorzystaniem środków świata realnego, czyli – realność jest źródłem wirtualności: jest to aspekt genetyczny. Ponadto realność dostarcza środków materialnych do tworzenia sfery wirtualnej, a jej likwidacja (w szczególności zniszczenie komputerów) prowadzi do unicestwienia obiektów wirtualnych. Ten aspekt zależności Gurczyński nazywa egzystencjalnym. Z kolei trzeci aspekt – funkcjonalny, polega na tym, że wszystkim procedurom obliczeniowym (stanowiącym bazę wirtualności) odpowiadają mechanizmy niższego rzędu (kierujące pracą komputerów). Nie wiemy przy tym, jak wspomniano wcześniej, jaka jest natura tych mechanizmów – cyfrowa czy analogowa.

Jeśli świat realny oparty jest na procesach cyfrowych, to istnienie rzeczywistości wirtualnej oznacza, że potrafimy odtwarzać świat, w którym żyjemy. Można też pokusić się o stwierdzenie, że wraz z rozwojem technologii owe wirtualne „kopie" fragmentów naszego świata będą przybliżały się coraz bardziej do oryginału. Jeśli natomiast przyjmiemy, że realność jest analogowa, to wirtualna rzeczywistość jest przykładem wytworzonej przez nas sfery, która w swej naturze jest zupełnie odmienna od świata realnego. Środowiska cyfrowe są bowiem skończone, podczas gdy analogowe są ciągłe i nieskończenie podzielne. Oznaczałoby to, zdaniem Gurczyńskiego, że po raz pierwszy w historii rozwój technologiczny doprowadził do powstania różnicy ontologicznej: „[…] tworząc środowiska cyfrowe, stworzyliśmy/odkryliśmy wcześniej nieznany rodzaj bytu, ontologicznie odmienny od tego wszystkiego, co znaliśmy i czego doświadczaliśmy. Oznaczałoby to, że tworząc rzeczywistość wirtualną, stworzyliśmy całkowicie nową ontologiczną przestrzeń, która funkcjonalnie jest/może być przestrzenią rzeczywistą" (s. 228).

Związki pomiędzy przestrzenią rzeczywistą a wirtualną dogłębnie analizował również Philip Brey; stworzył swoją ontologię obiektów wirtualnych, opartą na koncepcji rzeczywistości Searle'a.

4.6. Ontologia obiektów wirtualnych Breya

Philip Brey (1998, 2003) zaproponował ontologię obiektów wirtualnych, dzieląc je na dwa rodzaje: przedmioty tylko jednego rodzaju są w niej tak samo rzeczywiste jak ich odpowiedniki w rzeczywistym świecie. Tworząc swoją teorię, wykorzystał pochodzący od Searle'a (1998) podział rzeczywistości, ujęty w poniższym schemacie:



Rys. 8. Rzeczywistość według J. Searle'a

Pierwsza część rzeczywistości nie zależy od ludzkiej interpretacji, a jej elementy istnieją obiektywnie, na przykład Giewont, Wisła i tym podobne. Druga to rzeczywistość społeczna, której obiekty (takie jak śrubokręt, banknot czy ślub) są, przynajmniej w części, ustanowione przez konstrukcje i interpretacje społeczne. Obiekty ustanowione społecznie dzielą się z kolei na: (1) posiadające nieodłączne, obiektywne własności, powodujące, że pełnią one takie właśnie, a nie inne funkcje (śrubokręt i inne narzędzia), oraz (2) – te, które nie mają takich własności i których funkcja społeczna jest po prostu przydzielona (pieniądze). Obiekty pierwszego rodzaju muszą posiadać odpowiednie własności fizyczne, by wypełniać skojarzone z nimi funkcje. Tworzą one tak zwaną zwykłą rzeczywistość społeczną. Z kolei banknoty czy monety nie mają żadnych fizycznych cech potrzebnych do uznania ich za pieniądze, traktuje się je w określony sposób, ponieważ nałożono na nie taką funkcję – reprezentowanie wartości rynkowej – skojarzoną z nimi w obrębie danej instytucji, w tym przypadku w obrębie rynku. Takie obiekty tworzą rzeczywistość instytucjonalną.

W pierwszym zetknięciu z wirtualną rzeczywistością można odnieść wrażenie, że jej obiekty (przynajmniej większość z nich) są tego samego typu co przedmioty, z którymi spotykamy się w otaczającym nas świecie: drzewa, biurko, słowa, ołówki, rozmowy, wiadomości pisane, pieniądze i tym podobne. Zawiera ona jednak również takie obiekty, które nie mają swoich bezpośrednich fizycznych odpowiedników, na przykład kursor, plik czy fikcyjne postaci w grach komputerowych. Wszystkie obiekty napotykane w świecie wirtualnym, niezależnie od ich związków z rzeczywistością, Brey nazywa „obiektami wirtualnymi".

Obiekty rzeczywistości wirtualnej mają trzy podstawowe cechy: (1) nie istnieją fizycznie, nie mają masy i nie znajdują się w przestrzeni fizycznej, a więc przypominają obiekty fikcyjne[21] (takie jak postaci w powieściach czy filmach), (2) są interaktywne, można nimi operować, co często związane jest z ich „odpowiedziami" na nasze działania, oraz (3) mogą istnieć związki przyczynowe pomiędzy nimi a innymi obiektami. Wydaje się zatem, że mają one specjalny status ontologiczny – nie są obiektami fizycznymi, ale nie są również wyobrażeniami czy obiektami fikcyjnymi (Brey, 1998).

Wiele obiektów wirtualnych ma szczególny status również z innego powodu. Operacje bankowe na kontach internetowych, dokumenty elektroniczne, handel czy też wirtualne rozgrywki karciane lub szachowe są powszechnie akceptowaną częścią rzeczywistości. Nie można zatem twierdzić, że są one nierealne. Jednak z drugiej strony, jako symulacje (lub reprezentacje) obiektów rzeczywistych, mogą być nazywane nierealnymi lub

[21] Por. podrozdział 4.5.

sztucznymi. Wirtualne drzewa, woda czy skały nie są rzeczywiste, lecz w pewnym sensie sztuczne, a jako takie nie są elementami świata realnego. Zatem w rzeczywistości wirtualnej mamy do czynienia z dwoma różnymi typami obiektów. Istnieje bowiem różnica pomiędzy tymi, które są tylko symulacjami, a tymi, które uważane są za część rzeczywistości. Brey (2003) pisze:

> Będę nazywał obiektami wirtualnymi takie, które nie tylko symulują obiekty rzeczywistego świata, ale które są w każdy sposób równoważnymi im *ontologicznymi reprodukcjami* obiektów rzeczywistego świata. Zatem wirtualne wersje obiektów rzeczywistych są albo czystymi *symulacjami*, które są tylko podobne do obiektów świata rzeczywistego przez swoje cechy (własności) percepcyjne i interaktywne, albo ontologicznymi reprodukcjami, które mają swoje znaczenie dla świata rzeczywistego wykraczające poza dziedzinę środowiska wirtualnego (s. 277).

Rodzi się zatem pytanie, w jaki sposób odróżnić te dwa rodzaje obiektów i jak je scharakteryzować. Rozpatrując tę kwestię, Brey wykorzystuje opisaną powyżej ontologię Searle'a. Twierdzi, że rzeczywistość fizyczna oraz zwykła rzeczywistość społeczna (por. rys. 8) mogą być zazwyczaj tylko symulowane w środowiskach wirtualnych. Z kolei większość obiektów i faktów rzeczywistości instytucjonalnej, takich jak pieniądze i przeprowadzane na nich transakcje, są w nich „ontologicznie reprodukowane".

Obiekty fizyczne (takie jak drzewa, skały, góry) oraz obiekty zwykłej rzeczywistości społecznej (śrubokręt, krzesło i tym podobne) są jedynie symulowane w wirtualnej rzeczywistości. Ponieważ komputery nie mogą odtworzyć (wykonać reprodukcji) wszystkich własności fizycznych tych obiektów, które są kluczowe dla ich tożsamości – nie mogą być one w pełni „zreplikowane" przez obiekty wirtualne. Komputery mogą je jedynie symulować. Wirtualna skała nigdy nie będzie tak rzeczywista jak skała fizyczna, ponieważ cechy charakterystyczne skały (które ją w pewnym sensie definiują) to cechy fizyczne, obiektywne własności, a nie własności lub cechy społeczne. Takie wirtualne obiekty będą więc zawsze symulacjami, a nie ontologicznymi reprodukcjami.

Obiekty instytucjonalne natomiast, na przykład pieniądze czy własność prywatna, mogą istnieć „dosłownie" w rzeczywistości wirtualnej. Pieniądze wirtualne są tak samo realne jak papierowe, a przeprowadzane z nimi transakcje – tak samo realne jak te w rzeczywistości. Jest tak dlatego, że pieniądze wirtualne mają ten sam status, pełnią tę samą funkcję co rzeczywiste banknoty i monety. Wiąże się to z faktem, że obiekty instytucjonalne zyskują swój status ontologiczny poprzez nadanie im pewnej roli za pomocą funkcji statusu – to znaczy reguł postaci „*X* uznaje się za *Y* (w kontekście *C*)". Teoretycznie dowolna funkcja statusu może być skojarzona z dowolnymi

obiektami, jednak w praktyce jest ona nadawana tylko tym obiektom, które mają odpowiednie cechy. Okazuje się, że bardzo wiele obiektów wirtualnych dobrze nadaje się do nakładania na nie funkcji statusu. Dlatego też zdaniem Breya (2003)

> duża część rzeczywistości instytucjonalnej jest obecnie reprodukowana w środowiskach wirtualnych, gdzie dokonują się prawdziwe aktywności instytucjonalne takie jak kupno, sprzedaż, głosowanie, nabywanie własności, rozmowy, gry w szachy, hazard, kradzież, naruszenie własności prywatnej, testowanie, wstępowanie do klubu oraz gdzie można znaleźć odpowiednie do tego obiekty, takie jak kontrakty, pieniądze, litery oraz figury szachowe (s. 279).

Przyjrzyjmy się teraz temu, w jaki sposób obiekty instytucjonalne zyskują swój status i w jaki sposób użytkownicy mogą go rozpoznać. Rozpocznijmy od tej drugiej kwestii.

Rozpoznanie funkcji obiektów instytucjonalnych przez użytkowników wirtualnej rzeczywistości zależy od akceptacji dla danego obiektu odpowiedniej reguły: „*X* uznaje się za *Y* (w kontekście *C*)" oraz od zidentyfikowania go jako owego *X* (obiektu spełniającego regułę). Brey (2003) podaje następujący przykład. W pewnych środowiskach wirtualnych może istnieć reguła mówiąca o tym, że żółte fora dyskusyjne są tylko dla kobiet. Zatem aby rozpoznać dane forum jako tylko dla kobiet, trzeba znać regułę mówiącą, że żółte fora są właśnie takie, oraz zauważyć, że forum to ma kolor żółty.

Jednakże w świecie wirtualnym, podobnie jak w rzeczywistym, nie zawsze wygląda to tak prosto. Czasami ogląd samego obiektu nie wystarczy dla stwierdzenia jego statusu. W jaki sposób można na przykład rozpoznać, że dany mężczyzna jest żonaty? Identyfikację taką ułatwiają obiekty nazywane przez Searle'a wskaźnikami statusu; są one intencjonalnie przypisane (nadawane) innym obiektom w celu wskazania ich statusu. W przypadku żonatego mężczyzny takim wskaźnikiem jest obrączka ślubna.

Istnieje również inna trudność związana ze statusem obiektów instytucjonalnych w wirtualnej rzeczywistości. Ponieważ mogą one, jak wspomniano wcześniej, mieć status symulacji lub ontologicznej reprodukcji, użytkownik musi rozpoznać ten status jednoznacznie. Gdy naciskamy przycisk z napisem „wykonaj" pod formularzem przelewu, to musimy znać status ontologiczny tego przelewu. Przecież jeśli jest on ontologiczną reprodukcją, to nastąpi zmiana salda naszego (rzeczywistego) konta; jeśli natomiast jest on symulacją, na przykład – jest częścią gry polegającej na inwestowaniu, tak stać się nie musi. W przypadku braku wskaźnika statusu, chociażby w postaci odpowiedniego komunikatu, rozpoznanie tego statusu może być bardzo trudne lub nawet niemożliwe. Fakt ten wykorzystują często internetowi oszuści wyłudzający dane lub pieniądze.

Przejdźmy teraz do procesu nadawania obiektom instytucjonalnym ich statusu. Jest on dwustopniowy. Najpierw pewna grupa użytkowników – lub nawet pojedyncza osoba – proponuje nadanie danemu obiektowi nowego statusu, a następnie większość danej społeczności przyjmuje ją lub odrzuca.

Co decyduje o tym, że członkowie jakiejś grupy akceptują nową funkcję obiektu? Po pierwsze, proponujący ją człowiek lub instytucja może być pewnego rodzaju autorytetem. Gdy Narodowy Bank Polski emituje nowe banknoty, społeczeństwo bez zastanowienia przyjmuje je jako obowiązującą odtąd formę pieniądza, albowiem wprowadzająca je instytucja jest w tej sprawie autorytetem. Po drugie, proponowana funkcja może wydawać się użyteczna, na przykład neologizmy wchodzą do powszechnego użycia wtedy, gdy użytkownicy danego języka uznają je za potrzebne, gdyż w jakiś sposób ułatwiają one komunikację. Oczywiście, może się również zdarzyć kombinacja obu tych sytuacji.

Podobnie nadawany jest status obiektów instytucjonalnych w rzeczywistości wirtualnej. Funkcja statusu może zostać zaproponowana przez pewien autorytet, na przykład przez moderatora forum internetowego lub też przez grupę użytkowników, która uzna ją za użyteczną. I tak status żółtego forum „tylko dla kobiet" może zaproponować jego administrator lub też część użytkowników. Aby jednak funkcja ta zaczęła obowiązywać, potrzeba na to zgody większości osób, które z niego korzystają. Przyjmują one decyzję autorytetu bądź też uznają tę funkcję za użyteczną.

Podsumowując: Brey dzieli obiekty wirtualne na dwie kategorie – te, które są symulacjami obiektów rzeczywistych, oraz prawdziwe reprodukcje ontologiczne, będące obrazem rzeczywistości instytucjonalnej. Twierdzi on, że użytkownicy rozpoznają wirtualne obiekty jako instytucjonalne dzięki regułom typu „*X* uznaje się za *Y* (w kontekście *C*)", które są często ukazywane poprzez wskaźniki statusu (symbole, tekst, ikony lub przedmioty). Obiekty takie zyskują swój status instytucjonalny „odgórnie", poprzez autorytarne narzucenie go (przez autorytet), bądź też w wyniku zbiorowej decyzji jego użytkowników.

Tworząc ontologie obiektów wirtualnych, zarówno Gurczyński, jak i Brey określili związki pomiędzy realnością a wirtualnością. Związki owe są jednym z najczęściej dyskutowanych w literaturze zagadnień filozoficznych związanych z rzeczywistością wirtualną. Szczególnie interesująca wydaje się kwestia odróżnienia realności od fikcji; jest to jeden z najstarszych problemów filozoficznych. Przyjrzyjmy się teraz wybranym koncepcjom dotyczącym owego rozróżnienia w kontekście rzeczywistości wirtualnej.

4.7. Wirtualność a realność

Rozróżnienie tego, co pozorne, od tego, co rzeczywiste, jest zapewne jednym z podstawowych, najstarszych problemów metafizyki. Zagadki dotyczące obrazów lustrzanych czy też snów nurtują filozofów od dawna. Pojawienie się rzeczywistości wirtualnej stawia tę kwestię w nowym świetle, ujawnia też konieczność rewizji klasycznych stanowisk filozoficznych związanych z odróżnieniem rzeczywistości i pozoru.

W literaturze dotyczącej rzeczywistości wirtualnej spotyka się pogląd, iż obiekty wirtualne są pod wieloma względami podobne do odbić lustrzanych (por. np. Stanovsky, 2004). Wówczas kryterium rozróżnienia tego, co wirtualne, od tego, co realne, szukać należy w rozważaniach dotyczących odróżnienia obiektów rzeczywistych od ich obrazów w lustrze.

Zagadnienie odróżnienia przedmiotów rzeczywistych od ich lustrzanych obrazów filozofowie rozważają od bardzo dawna, zdecydowanie dłużej niż istnieje rzeczywistość wirtualna. Już Plotyn (204-269) w *Enneadach* (por. np. Plotyn, 1959) studiował problem odbić lustrzanych – twierdził on, że w większości przypadków łatwo jest odróżnić odbicie od realnego przedmiotu. Zgadzał się jednak, że w pewnych warunkach (jeśli odbicie w lustrze jest trwałe, a samego lustra nie widać) odbicia te mogą nas zwodzić.

Rzeczywiście, dość łatwo jest odróżnić obiekt od jego obrazu w lustrze. Odbicia takie są bowiem przemijające, przelotne, tymczasowe, nietrwałe w czasie, a czasami nawet niespójne z pozostałymi wrażeniami percepcyjnymi. Ponadto w większości przypadków lustro nie pozostaje niezauważone, widoczna jest jego struktura, która odbija światło, oraz ramy. Co więcej, obiekty w lustrze przedstawiane są zawsze na płaszczyźnie – a więc ich dotknięcie na ogół umożliwia odróżnienie obiektu od jego odbicia.

Czy obiekty rzeczywistości wirtualnej, będące pewnego rodzaju obrazami (odbiciami) realnego świata, można porównać do odbić lustrzanych? Czy relacja, w jakiej pozostają one w stosunku do obiektów rzeczywistych, jest prostym przeniesieniem związków pomiędzy obrazami lustrzanymi a ich realnymi pierwowzorami? Wydaje się, że wirtualna rzeczywistość jest bardziej skomplikowana niż obrazy lustrzane. Obiekty w niej prezentowane nie są zazwyczaj ograniczone do dwuwymiarowej grafiki, jak to się dzieje w przypadku obrazu odbitego w lustrze, nie są też one nietrwałe i przemijające – mogą trwać tak samo długo (a czasem nawet dłużej), jak przedmioty czy wydarzenia rzeczywiste. Co więcej, lustro odbija jedynie istniejące obecnie przedmioty, podczas gdy obiekty w wirtualnej rzeczywistości mogą nie mieć swoich realnych odpowiedników, mogą istnieć tylko w niej (jak na przykład fikcyjne postaci w grach komputerowych). Rzeczywistość wirtualna jest zatem zupełnie nową rzeczywistością, często bez jakiegokolwiek

odniesienia do realności i dlatego konieczne jest określenie dla niej nowych – innych niż w przypadku odbić lustrzanych – metod rozróżniania tego, co jest rzeczywiste (autentyczne), od tego, co wirtualne (pozorne lub nieautentyczne).

Ludzie doświadczający wirtualnej rzeczywistości opisując swoje doznania, często porównują je do snu. Sny są bowiem tak samo „zanurzające" jak światy wirtualne. Wielu filozofów na przestrzeni dziejów rozważało sposoby odróżnienia snu od jawy, szukając takich, które mogą być pomocne w odróżnieniu tego, co rzeczywiste, od tego, co wirtualne.

Jednym z filozofów piszących na temat snów był Kartezjusz, który w dziele *Medytacje o pierwszej filozofii* wśród wielu poruszanych tematów analizował również kwestię odróżnienia snu od jawy. Zdawał on sobie sprawę z tego, że może nie istnieć kryterium pozwalające na takie rozróżnienie. W ostatniej ze swych medytacji (2010) sformułował pewne wskazówki dotyczące tego zagadnienia:

> Teraz bowiem zdaję sobie sprawę, jak bardzo wielka jest między nimi różnica, polegająca na tym, że nigdy pamięć nie potrafi tak powiązać snów z wszystkimi naszymi zdarzeniami życia, jak wiąże z nimi to, co się zdarza na jawie. Bo rzeczywiście, gdyby mi się ktoś nagle ukazał i natychmiast znikł, jak to się zdarza w snach, tak oczywiście, bym nie widział ani skąd przyszedł, ani dokąd poszedł, to słusznie mógłbym go uważać raczej za widmo, czy też za wytwór fantazji powstały w moim mózgu, niż za prawdziwego człowieka. Gdy jednak zjawiają mi się takie rzeczy, co do których wyraźnie zauważam, skąd, gdzie i kiedy przychodzą do mnie i których ujęcie wiążę bez żadnej luki z całym moim pozostałym życiem, wtedy jestem całkowicie pewny, że zjawiają mi się one na jawie, a nie we śnie (s. 92).

Kartezjusz zatem uważa, że można odróżnić sen od jawy, badając spójność konkretnego naszego doświadczenia z pozostałymi zdarzeniami z życia. Jednak nie zawsze jest to kryterium rozstrzygające, ponieważ sny bardzo często są spójne i nie ma w nich sytuacji takich jak te opisywane powyżej, pozwalających na stwierdzenie, że śnimy. Rzeczywistość wirtualna bez trudu przejdzie zaproponowany przez Kartezjusza test jasności i spójności. Nie można zatem za jego pomocą odróżnić świata rzeczywistego od wirtualnego. Nie wydaje się więc, by porównywanie wirtualności do snu było prawomocne.

Jeszcze wyraźniej widać różnicę pomiędzy snem a wirtualną rzeczywistością, gdy popatrzymy na nie z perspektywy podmiotu (śniącego lub uczestnika rzeczywistości). Marzenia senne są bowiem doznaniem czysto subiektywnym, są doświadczeniami osobistymi człowieka. Doświadczanie wirtualnej rzeczywistości natomiast jest intersubiektywne, podobnie jak doświadczanie świata rzeczywistego.

Zatem uznanie porównania wirtualności do snu – a jej obiektów do odbić lustrzanych – za kryterium odróżniania świata rzeczywistego od wirtualnego wydaje się niewłaściwe. Zbyt wiele bowiem różni obiekty wirtualne od obrazów lustrzanych i marzeń sennych. Plotyn i Kartezjusz uważali trwałość i spójność za kryteria rzeczywistości, a przemijalność – za oznakę pozoru. Jednakże rozróżnienie takie nie ma zastosowania w przypadku rzeczywistości wirtualnej. Jej obiektom zdaje się przysługiwać zupełnie inny rodzaj istnienia niż snom, odbiciom lustrzanym i innym „zwykłym" reprezentacjom[22].

Można szukać sposobów odróżnienia tego, co wirtualne, od tego, co realne, z ograniczeniem się do pewnego fragmentu lub rodzaju rzeczywistości wirtualnej, na przykład do gier jako typowego jej przykładu.

Łukasz Piwek w swojej pracy *The Dynamic Representation of Reality and of Ourself between Real and Virtual Worlds* (2008) analizuje gry komputerowe, czyli w szerokim rozumieniu wszystkie formy gier oparte na aktywnej rzeczywistości wirtualnej, w których podmiot ma rozmaite cele, misje i dążenia (na przykład Word of Warcraft[23] czy The Sims[24]). Mianem „pasywnej rzeczywistości wirtualnej" określa natomiast te środowiska, w których podmiot takich celów nie posiada (na przykład surfowanie po Internecie w ogóle).

Podmiot może działać w dwóch rzeczywistościach: otwartej (świat zewnętrzny) i zamkniętej (świat wirtualny, którego doświadcza podczas grania w wirtualne gry). Tym, co je odróżnia, jest zdaniem Piwka (2008) stopień złożoności, definiowany jako „bogactwo bodźców oraz liczba dostępnych opcji działania" (s. 26). Stopień złożoności rzeczywistości zamkniętej wydaje się mniejszy niż rzeczywistości otwartej, ponieważ dostarcza ona mniej bodźców zmysłowych, a możliwości działania w niej są ograniczone. Przyczyną tego jest fakt, że podmiot ma dostęp do niej tylko poprzez interfejs pośredni. Świat zewnętrzny natomiast jest dostępny bezpośrednio; jedynym „filtrem" wpływającym na jego doświadczanie przez podmiot jest mózg i system nerwowy podmiotu.

---

[22] Przekonanie o specjalnym statusie ontologicznym obiektów wirtualnych nie jest powszechne. Na przykład Derek Stanovsky (2004) twierdzi, że w świetle metafizyki Kanta można je traktować, w mniejszym lub większym stopniu, tak samo jako inne (zwykłe) obiekty ludzkiego doświadczenia. Co więcej, twierdzi on, że dla Kanta każde doświadczenie jest w pewnym sensie „wirtualne".

[23] World of Warcraft to seria gier przygodowych, w których użytkownicy przenoszą się w świat Azeroth, gdzie toczy się konflikt pomiędzy Przymierzem a Hordą; uczestnikiem jego staje się użytkownik gry. Celem gry jest pokonanie obozu przeciwnego.

[24] Por. przypis 14.

Jedną z podstawowych cech rzeczywistości zamkniętej (wirtualnej) jest łatwość jej percepcyjnego uporządkowania; rzeczywistość otwarta jest tej własności pozbawiona. Podmiot zazwyczaj preferuje systemy prostsze – a rzeczywistość gier jest takim właśnie prostym systemem. Działając w niej, podmiot ma jasno zdefiniowane preferencje i cele działania oraz wzorce postępowania. Być może jest to jedna z cech rzeczywistości wirtualnej wpływająca w znacznym stopniu na jej wielką popularność.

Warto w tym miejscu zauważyć, że światy wirtualne i rzeczywistość nie są oderwane, lecz wpływają na siebie nawzajem. Z jednej strony, w świecie wirtualnym symuluje się obiekty i wydarzenia świata rzeczywistego. Z drugiej – światy wirtualne mają coraz większy wpływ na rzeczywistą ekonomię, kulturę, politykę i edukację. Wpływy te widoczne są również w przypadku gier komputerowych. W grach takich jak Second Life[25], Entropia Universe[26] czy Word of Warcraft gracze kupują i sprzedają wirtualne posiadłości, budują domy i inne elementy rzeczywistości oraz sprzedają towary na internetowych aukcjach za prawdziwe (rzeczywiste) pieniądze. Tak więc działania w świecie wirtualnym powodują zmiany w rzeczywistości (w faktycznym stanie posiadania graczy). Co więcej, wirtualny świat gier wykorzystywany jest do różnego rodzaju „rzeczywistych" badań, na przykład do analizy zachowań ludzi podczas epidemii i klęsk żywiołowych (w grze Word of Warcraft[27]) czy też – do badań marketingowych, w których znane marki testują popyt na swoje towary.

Analiza związków świata wirtualnego i rzeczywistości zaprezentowana przez Piwka ograniczona jest do zagadnień dotyczących gier komputerowych. Potrzebne są jednak dalsze, bardziej ogólne badania tych związków. Wydaje się, że dotychczasowe studia filozoficzne dotyczące wirtualnej rzeczywistości nie są kompletne, a ten nowy świat – świat wirtualny, stale dostarcza filozofom nowych tematów do przemyśleń.

---

[25] Second Life to wirtualny świat, w którym specjalne narzędzie umożliwiają użytkownikom (nazywanym mieszkańcami) jego modyfikację oraz uczestnictwo w wirtualnej gospodarce. Istnieje w nim specjalna wirtualna waluta (Linden Dollars), którą można zakupić za rzeczywiste pieniądze.

[26] Entropia Universe to wirtualny świat z własną gospodarką wewnętrzną i walutą o stałym kursie (PED – Project Entropia Dollars). Gracze za pośrednictwem swoich postaci (awatarów) tworzą wirtualne społeczeństwo. Entropia łączy w sobie elementy gier, a także systemów hazardowych. Przedmioty wirtualne nabyte w Entropii mają rzeczywistą wartość pieniężną. Każdy gracz może w dowolnym czasie przeprowadzić wymianę zgromadzonych w grze środków na główne waluty światowe (m.in. USD, EUR, GBP) według stałego kursu i ich wypłatę.

[27] Badania takie prowadzi Tufts University (prywatny uniwersytet badawczy w Medford/Somerville, niedaleko Bostonu).

4.8. Podsumowanie

Zjawisko nazywane rzeczywistością wirtualną definiuje się zazwyczaj na dwa sposoby. Gdy bierzemy pod uwagę aspekt psychologiczny, czyli przeżycia podmiotu jej doświadczającego, możemy stwierdzić, że nie jest ona zjawiskiem nowym, gdyż doświadczenia związane z generowaną komputerowo rzeczywistością podobne są do tych, z którymi mamy do czynienia podczas obcowania z literaturą i sztuką. Gdy jednak pojmujemy świat wirtualny jako nieodłącznie związany z technologią komputerową, przyznać musimy, że jest to fenomen zupełnie nowy, gdyż wkraczając do świata gry komputerowej lub do Internetu, porzucamy realność i zagłębiamy się w wirtualność. W rozdziale tym przeanalizowano cechy rzeczywistości wirtualnej oraz wybrane zagadnienia ontologiczne z nią związane.

Opisując wirtualność, wskazuje się na całą gamę jej cech i zjawisk z nią związanych. Niektóre z omówionych w podrozdziale 4.2 cech rzeczywistości wirtualnej to pojęcia nowe (takie jak komunikacja sieciowa czy hipertekst), inne są tradycyjnym tematem badań filozofii, nauki o kulturze i psychologii (symulacja, fikcjonalizacja oraz czas). Wyniki analiz pokazują, że wiele z nich to pojęcia wieloznaczne, a różne ich rozumienie prowadzi do innego spojrzenia na świat wirtualny. Trudno jest zatem podać zbiór cech jednoznacznie go definiujących.

Tworząc ontologię wirtualności, a w szczególności określając sposób istnienia obiektów wirtualnych, można wykorzystać jej związek z dobrze określonymi filozoficznie pojęciami symulacji oraz fikcji.

Obiekty wirtualne są bardzo często traktowane jako symulacje (przybliżenia) obiektów rzeczywistych (prawdziwych). Materialiści mogą na przykład twierdzić, że wirtualność jest wytworem komputera (jego obwodów, kabli i oprogramowania). Kuszące wydaje się również ujmowanie rzeczywistości wirtualnej jako potwierdzenia metafizyki Platona, w której świat idei jest realizowany w mniej doskonałym świecie rzeczy – podobnie jak świat wirtualny jest niedoskonałym odbiciem świata rzeczywistego. Można również widzieć wirtualność jako dzieło człowieka, którego pierwotnym źródłem jest rzeczywistość empiryczna lub świat obiektów abstrakcyjnych, a wtórnym – ludzki umysł (por. podrozdział 4.3).

Ważnym aspektem ontologicznym obiektów wirtualnych jest ich intencjonalność. Świat wirtualny istnieje, ponieważ człowiek chce obdarzyć go istnieniem. W kontekście wirtualnej rzeczywistości wśród obiektów intencjonalnych na szczególną uwagę zasługują obiekty fikcyjne. W podrozdziale 4.4 przeanalizowano własności tych obiektów. Niektóre z nich, takie jak: intersubiektywność, niezupełność ontologiczna oraz dwupoziomowość określenia, przysługują również obiektom wirtualnym. Istnieją jednak takie

fundamentalne cechy obiektów wirtualnych, których nie posiadają – fikcyjne: dynamiczność, interaktywność oraz ufundowanie na procesach cyfrowych. Nie można zatem utożsamiać tych dwóch rodzajów obiektów.

W ontologii rzeczywistości wirtualnej Gurczyńskiego, przedstawionej w podrozdziale 4.5, istotną rolę odgrywa rozróżnienie pojęcia „realności" (związanej ze światem materialnym, zewnętrznym) oraz „rzeczywistości" rozumianej jako fizyczna przestrzeń, w której mogą zachodzić ucieleśnione działania służące rozwiązywaniu problemów. Podejmując takie działania, podmiot napotyka opór – wynikający z praw zachodzących w danym środowisku. W takim ujęciu można rozumieć świat wirtualny jako pewnego rodzaju rzeczywistość.

Gurczyński proponuje traktowanie rzeczywistości wirtualnej jako przestrzeni, z jej ograniczeniami fizycznymi, w której mogą zachodzić ucieleśnione działania napotykające na opór, a służące rozwiązywaniu problemów. Rozważa on również trzy aspekty zależności pomiędzy światami wirtualnym i realnym: genetyczny (realność jest źródłem wirtualności), egzystencjalny (likwidacja realności powoduje unicestwienie wirtualności) oraz funkcjonalny (wszystkim procedurom obliczeniowym stanowiącym bazę wirtualności odpowiadają mechanizmy niższego rzędu, kierujące pracą komputerów).

Z analizy związków pomiędzy wirtualnością i realnością wyrosła ontologia obiektów wirtualnych, przedstawiona w podrozdziale 4.6. Jej twórca Philip Brey mianem „obiektów wirtualnych" określa wszystkie obiekty napotykane w świecie wirtualnym, niezależnie od tego, czy mają one swoje odpowiedniki w rzeczywistości, czy też nie. Analizując sposób ich istnienia, dzieli obiekty te na dwa rodzaje: (1) takie, które są wyłącznie symulacjami obiektów rzeczywistych, oraz (2) reprodukcje ontologiczne, które mają wpływ na świat realny. Twierdzi, że obiekty rzeczywistości fizycznej (drzewa, skały, góry) oraz zwykłej rzeczywistości społecznej (śrubokręt, krzesło) w wirtualnym świecie mogą być jedynie symulowane. Większość obiektów i faktów rzeczywistości instytucjonalnej natomiast (pieniądze, własność prywatna) może mieć w nim swoje ontologiczne reprodukcje.

Mówiąc o rzeczywistości wirtualnej, najczęściej traktuje się ją jako pojęcie opozycyjne względem „rzeczywistości realnej", co stanowi nawiązanie do jednego z najstarszych problemów ontologicznych – kwestii odróżnienia tego, co realne, od tego, co pozorne (nierzeczywiste). Można szukać jej rozwiązania, porównując wirtualność do snu, a obiekty wirtualne – do odbić lustrzanych. Przedstawione w podrozdziale 4.7 analizy pokazują jednak, że nie jest to podejście właściwe.

Niektórzy badacze odrzucają rozumienie wirtualności jako pojęcia opozycyjnego wobec realności. Na przykład Krystyna Wilkoszewska we wpro-

wadzeniu do książki *Estetyka wirtualności* (2005) proponuje traktowanie wirtualności jako zjawiska transwersalnego, ponieważ w codziennych ludzkich doświadczeniach realność i wirtualność przenikają się. Twierdzi ona, że prefiks „trans" podkreśla rozumienie rzeczywistości wirtualnej jako pewnego rodzaju „dodatku" do realnego świata, oraz uwypukla fakt, że „wirtualne modyfikuje realne, a to, co realne, kształtuje to, co wirtualne" (s. 9).

W rozdziale tym przedstawiono tylko wybrane idee związane w ontologią rzeczywistości wirtualnej. Ciekawe może okazać się zbadanie – w kontekście wirtualności – innych niż zaprezentowane powyżej tradycyjnych koncepcji filozoficznych, takich jak metafizyka Heideggera (rozważa się w niej obiekty zawsze w powiązaniu z ich doświadczaniem) czy też teoria symulacji i hiperrzeczywistości Jeana Baudrillard'a, który twierdził, że powstanie rzeczywistości wirtualnej oznacza kres naszej zdolności rozróżnienia pomiędzy rzeczywistością a pozorem. Z kolei do opisu doświadczania rzeczywistości wirtualnej odpowiednie wydaje się podejście fenomenologiczne, koncentrujące się na opisie i oglądzie tego, co bezpośrednio dane[28].

Rzeczywistość wirtualna stawia też w nowym świetle klasyczne problemy filozoficzne, niezwiązane z nią bezpośrednio, jak na przykład kwestia relacji pomiędzy umysłem a ciałem. Heidt (1999) twierdzi, że powstanie światów wirtualnych podkreśla nierozłączność umysłu i ciała oraz konieczność ucieleśnienia wszelkich form ludzkiego poznania i wiedzy. Nawet w bezcielesnym świecie rzeczywistości wirtualnej muszą istnieć pewnego rodzaju wirtualne ciała, umożliwiające ludziom interakcję z nim.

Wraz z powstaniem wirtualnej rzeczywistości nowy wymiar zyskało również pojęcie tożsamości podmiotu, własnego „ja". Ten szczególnego rodzaju świat otwiera możliwość równoczesnego istnienia wielu tożsamości danego podmiotu. Stworzenie ich i utrzymywanie możliwe jest dzięki rozmaitym narzędziom: od graficznych awatarów, reprezentujących użytkowników w środowiskach wirtualnych, poprzez pseudonimy wykorzystywane w komunikacji na forach dyskusyjnych, aż po możliwość posiadania wielu kont e-mailowych. Tożsamość w wirtualnej rzeczywistości jest też bardziej plastyczna niż w realnym świecie: można ją zmieniać, edytować, eliminować lub tworzyć tożsamości fikcyjne (Stanovsky, 2004). A zatem pojęcie tożsamości, podobnie jak przestrzeni i czasu, zyskało w tejże rzeczywistości nowe cechy.

Miczka (2009) zwraca uwagę, że wirtualna rzeczywistość ma jeszcze jedną, bardzo ciekawą cechę – nie istnieje fizycznie, a jednak wpływa na

---

[28] Podejście fenomenologiczne różni się od nastawienia naturalnego bezzałożeniowością. W nastawieniu naturalnym mamy na temat świata pewne założenia, domysły, teorie, spekulacje. Fenomenologia nawołuje do ich odrzucenia po to, by przyjrzeć się światu takiemu, jakim się on jawi.

człowieka podobnie jak rzeczywistość, wywołując w nim procesy mentalne i fizyczne. A więc „jeśli ten [wirtualny] świat nie istnieje, ale istnieją jego skutki, to można chyba mówić o istnieniu ‘przesunięcia ontologicznego’?” (s. 24, przyp. mój – I. B-K.).

Pojawienie się rzeczywistości wirtualnej zapoczątkowało też powstanie wielu nowych kwestii filozoficznych, zarówno ontologicznych, jak i epistemologicznych czy etycznych z nią związanych. Coraz bliższy związek świata wirtualnego z rzeczywistym podkreśla natomiast konieczność stworzenia odpowiedniej ontologii dla wirtualności, która jest przecież istotną częścią naszej codzienności.

Zakończenie

Celem tej książki była próba zdefiniowania informatyki jako nauki przez pryzmat badanych przez nią obiektów. Zawarte w niniejszej publikacji rozważania prowadzą do wniosku, że zdefiniowanie informatyki jako nauki o *X* wiąże się w każdym przypadku z problemami, bowiem przedmioty te (*X*-y) nie zawsze są precyzyjnie określone. Każdy z zaprezentowanych poglądów dotyczących obiektu badań informatyki pociąga za sobą konieczność rozwiązania wielu kwestii związanych z ich naturą. Niełatwo jest zdefiniować komputer. Równie trudno jest opisać naturę wykonywanych na nim programów i określić wzajemne związki pomiędzy pojęciami związanymi z procesem ich tworzenia, takimi jak: algorytm, procedura, specyfikacja, program czy implementacja. Szczególnych trudności przysparza charakterystyka wieloznacznego pojęcia „informacja", nawet jeśli ogranicza się jego rozumienie tylko do informatyki. Kłopoty pojawiają się również w związku z opisem rzeczywistości wirtualnej.

Rozdział pierwszy poświęcono próbom odpowiedzi na pytanie o naturę komputerów, gdyż informatyka w potocznym rozumieniu to nauka o urządzeniach nazywanych komputerami. Z poglądem tym zgadza się również część informatyków, choć ich rozumienie terminu „komputer" istotnie różni się od rozumienia potocznego. Czym zatem jest komputer i jaka jest jego natura? W rozdziale tym przeanalizowano zagadnienia filozoficzne związane z dwojakim rozumieniem komputera: jako urządzenia oraz jako obiektu abstrakcyjnego.

Najczęściej pod pojęciem „komputer" rozumie się samodzielne urządzenie fizyczne lub też jego część. Mianem tym określa się zarówno laptopy, serwery, netbooki, jak i systemy wbudowane w telefony, samochody, sprzęt gospodarstwa domowego, a nawet w statki kosmiczne. Ze względu na wielość i różnorodność obiektów określanych tym terminem precyzyjne jego zdefiniowanie jest zadaniem bardzo trudnym. Przeprowadzone analizy historyczne pokazują, że nie sposób nawet określić, którą z powstałych na

przestrzeni kilku stuleci maszyn można nazwać pierwszym „komputerem", i co za tym idzie – nie jest możliwe wskazanie jego wynalazcy. Również analizy współczesnych jego definicji prowadzą do wniosku, że nie ma jednego obowiązującego jego określenia, a sami informatycy nie są zgodni co do tych cech komputera, które wyróżniają go spośród innych urządzeń.

Komputer jest narzędziem stworzonym przez człowieka w wyniku celowych działań – jest on zatem pewnego rodzaju artefaktem – artefaktem technicznym. Można więc szukać rozwiązania kwestii filozoficznych związanych z jego naturą pośród ontologii artefaktów technicznych. Wśród filozofów powszechne jest przekonanie o dualnej naturze takich artefaktów, są one bowiem zarówno intencjonalne (jako realizacje ludzkich celów i zamiarów), jak i fizyczne (ich budowa umożliwia realizację pewnych funkcji). Wydaje się jednak, że dotychczas nie powstała żadna ogólnie akceptowana ontologia artefaktów technicznych, która łączyłaby te dwa nieodłączne ich aspekty, a największą trudnością istniejących teorii jest zdefiniowanie pojęcia ich funkcji.

W rozdziale pierwszym przedstawiono podstawowe koncepcje funkcji artefaktów technicznych, w tym dwie najważniejsze: teorie przyczynowe oraz intencjonalne. Wyniki przeprowadzonych analiz wskazują na liczne trudności wynikające z przyjęcia każdej z nich. Teorie przyczynowe zakładają, że funkcję obiektów wyznaczają ich aktualne możliwości fizyczne. Takie ujęcie jest z jednej strony zbyt liberalne (gdyż nie nakłada wystarczających ograniczeń na funkcję), a z drugiej – zbyt wąskie (ponieważ koncentruje się jedynie na *aktualnych* możliwościach fizycznych przedmiotu). Najpoważniejszą jednak wadą teorii przyczynowych jest brak kryteriów poprawnego działania artefaktu. Do stwierdzenia, czy urządzenie działa poprawnie, nie wystarczy analiza jego aktualnych cech fizycznych – konieczne jest odwołanie się do intencji projektanta. Dobrze ujmują to teorie intencjonalne, w których przyjmuje się, że to agent przypisuje funkcję do artefaktu, pełniącego określoną funkcję tylko wtedy, gdy realizuje cel. Teorie te pozwalają zatem na sformułowanie kryterium poprawnego działania artefaktów, ale nie określają sposobu, w jaki funkcja jest „przydzielana" obiektowi, ani też nie wyjaśniają stosunku pomiędzy fizyczną substancją tego obiektu a jego funkcją. Ciekawą alternatywą dla teorii przyczynowych i intencjonalnych są, stosowane głównie w biologii, etiologiczne teorie funkcji przypisujące obiektom funkcje, bazując na ich historii przyczynowej. Co prawda, nie powstała jeszcze etiologiczna teoria funkcji dla artefaktów technicznych – ze względu na brak dla nich dobrze rozwiniętej i powszechnie akceptowanej teorii przyczynowej – ale sformułowano jej podstawowe założenia.

Budowę ontologii artefaktów technicznych należy rozpocząć od sformułowania kryteriów jej adekwatności. W rozdziale pierwszym przeanalizo-

wano dwa takie kryteria dla teorii artefaktów technicznych: niedookreślenie oraz ograniczenia realizowalności – odpowiadające w pewnym sensie ich dualnej naturze. Z pierwszym z nich – postulującym uwzględnienie faktu, że relacja pomiędzy artefaktem i jego funkcją nie jest relacją jeden do jeden – zgadza się większość badaczy; jednak kryterium drugie – informacja o artefakcie niesie pewną informację o jego materialnej bazie i *vice versa* – jest szeroko dyskutowane. Analizy sposobów rozumienia relacji pomiędzy artefaktem i jego materialną bazą: superweniencji oraz ukonstytuowania ukazują, że żaden z nich nie jest odpowiedni w przypadku artefaktów technicznych. Nie dziwi zatem fakt, że dotychczas nie powstała żadna powszechnie akceptowana teoria, którą można by zaadaptować na potrzeby ontologii urządzeń nazywanych „komputerami".

Rozumienie komputera jako urządzenia fizycznego determinuje dopuszczalne metody sprawdzania jego poprawnego działania. Najczęściej wykorzystuje się w tym celu testowanie – empiryczne sprawdzenie urządzenia. Jednakże istnieją też, przeanalizowane w rozdziale pierwszym, próby stworzenia formalnych dowodów poprawnego działania systemów komputerowych. Wiele zaprezentowanych argumentów świadczy na rzecz tezy, że właściwe działanie urządzeń nazywanych komputerami jest zawsze kwestią empiryczną i ma niewiele wspólnego z takimi pojęciami, jak „dowód formalny" czy „ścisła argumentacja". Nie jest to jednak jedyne ujęcie pojęcia „komputer", można je bowiem rozumieć zupełnie inaczej – jako obiekt abstrakcyjny.

Pojęcie komputera jako maszyny abstrakcyjnej zostało wprowadzone przez logików na potrzeby badań nad naturą obliczalności. Istnieje wiele abstrakcyjnych – matematycznych – precyzacji pojęcia „komputer" (jednym z nich jest „maszyna Turinga"). Wszystkie one definiują komputer w terminach czysto syntaktycznych – jako wykonujący operacje na symbolach. W podrozdziale 1.2 przeanalizowano filozoficzne konsekwencje takiego ujęcia, w szczególności problem uniwersalnej realizowalności: każdy obiekt fizyczny jest cyfrowym komputerem, gdyż zawsze można znaleźć taki jego syntaktyczny opis, zgodnie z którym obiekt ten jest realizacją jakichś abstrakcyjnych bytów formalnych. Trudności związane z rozumieniem komputera jako obiektu matematycznego były dla badaczy jedną z motywacji do poszukiwań innych jego charakterystyk, omówionych w podrozdziale 1.3.

Każda z przedstawionych prób określenia komputerów jako obiektów zainteresowań informatyki związana jest z wieloma trudnościami. Próby ich rozwiązania prowadzą jednakże do głębszego zrozumienia nie tylko natury samych komputerów, ale również innych związanych z nimi pojęć, takich jak: artefakt, obliczalność, algorytm czy maszyna abstrakcyjna. Komputery stawiają zatem przed filozofami coraz to nowe, liczne wyzwania.

Ciekawym z punktu widzenia filozofii zagadnieniem związanym bezpośrednio z komputerami jest rozróżnienie terminów „hardware" i „software". Z pozoru sprawa jest prosta – hardware to fizyczne maszyny (komputery), a software – oprogramowanie, które powoduje, że działają one w określony sposób. Jednak przeprowadzone analizy pokazują, że rozróżnienie takie nie jest wystarczające, gdyż nie można sprowadzić go jedynie do fizycznych cech urządzeń, których programy nie posiadają. Istnieją bowiem programy wbudowywane bezpośrednio w urządzenia, stanowiące ich nieodłączną część. Co więcej, postęp technologiczny powoduje, że pewne operacje wcześniej obsługiwane programistycznie (stanowiące software), obecnie realizowane są sprzętowo (stając się hardware). Różnic pomiędzy software a hardware należy zatem szukać poza ich cechami fizycznymi. Można twierdzić na przykład, że hardware jest programowalny, a software przenośny albo też – że hardware jest konkretny, a software abstrakcyjny. W podrozdziale 1.1 podjęto również próbę zdefiniowania terminów „software" i „hardware" jako odrębnych, jednoznacznie zaklasyfikowanych bytów, w obrębie *Basic Formal Ontology*. Zazwyczaj definiuje się je jako pojęcia opozycyjne, choć dychotomia software/hardware jest często krytykowana jako czysto pragmatyczna. Nie ulega jednak wątpliwości, że pojęcia „komputer" i „program" są ściśle ze sobą związane i w pewnym sensie dopełniają się. Szukając odpowiedzi na pytanie o przedmiot badań informatyki, nie można zatem pominąć programu komputerowego.

W rozdziale drugim przedstawiono zagadnienia filozoficzne związane z pojęciem programu komputerowego, rozpoczynając od prób jego zdefiniowania. Chociaż wydaje się, że informatycy doskonale rozumieją, czym jest program, to podanie jego ścisłej definicji czy też wskazanie podstawowych cech może przysparzać trudności. Najczęściej podawane określenia obiektu nazywanego „programem" koncentrują się na opisie jego wpływu na działanie komputera, na przykład szkolne definicje stwierdzają, że jest to ciąg instrukcji dla komputera (a właściwie – procesora), określających jego zachowanie podczas wykonania programu. Przeprowadzone analizy pokazują, że jest to zbyt daleko idące uproszczenie.

Można twierdzić, że program komputerowy jest „konkretną abstrakcją" posiadającą nośnik zapisu (który jest abstrakcją) oraz nośnik wykonania (konkretną realizację w półprzewodnikach), dzięki któremu program komputerowy może zostać wykonany przez maszynę. Stajemy jednak wówczas przed pytaniem o to, w jaki sposób obiekt może być jednocześnie abstrakcyjny i konkretny. Odpowiedzi na nie można szukać po sprowadzeniu kwestii relacji pomiędzy konkretem a abstrakcją do klasycznego problemu filozoficznego – związku duszy z ciałem. Wtedy – zgodnie z doktryną monizmu – program jest jednym bytem, a abstrakcja i konkret to tylko jego aspekty, bądź też

– w duchu dualizmu – jest on obiektem zarówno abstrakcyjnym, jak i konkretnym. Innym rozwiązaniem jest poszukiwanie wyjaśnienia dualnej natury programów komputerowych w filozofii muzyki. Utwór muzyczny ma bowiem wiele wspólnego z programem komputerowym: jego zapis (partyturę) można porównać do kodu źródłowego programu (ciągu instrukcji), a wykonanie utworu na instrumentach – do uruchomienia programu na komputerze. Odpowiednie wydaje się zatem określenie programów – kolejnymi, po utworach muzycznych – przykładami przedmiotów quasi-partykularnych (które nie powstają w procesie abstrakcji z partykulariów), zapisywanych w postaci kodu źródłowego i realizowanych przez wykonania na komputerach.

Zupełnie inaczej widzą program komputerowy zwolennicy tak zwanego paradygmatu matematycznego w informatyce. Twierdzą oni, że programy te są obiektami matematycznymi. Z poglądem tym wiąże się wiele pytań. Czy programy są przykładami dobrze znanych bytów matematycznych (na przykład funkcjami), czy też tworzą odrębną grupę? Jaka jest ich natura?

Programy komputerowe można definiować poprzez odwołanie się do obiektu abstrakcyjnego, jakim jest cyfrowy wzorzec. Jednak samo pojęcie „wzorca" wydaje się niejasne, a określenie programu jako cyfrowego wzorca związane jest z przyjęciem wielu silnych założeń ontologicznych, którym brak pełnego uzasadnienia. Trudno jest również, przyjmując taką definicję programów, sformułować kryterium ich odróżniania.

Określenie jasnego sposobu identyfikacji programów komputerowych jest jednym z podstawowych zagadnień ontologicznych z nimi związanych i ma także dalekosiężne konsekwencje – nie tylko filozoficzne, ale również praktyczne, związane chociażby z ich ochroną prawną. Najłatwiej jest utożsamić program z jego fizyczną reprezentacją (na przykład z zapisem w języku programowania). Prowadzi to jednak do uznania, że zarówno programy o kodzie różniącym się tylko wyglądem czcionki, jak i działające w ten sam sposób – ale zapisane w różnych językach programowania, należy zawsze traktować jako różne, co przeczy uznawanej powszechnie praktyce. Określenia kryterium identyczności programów należy zatem szukać poza ich cechami fizycznymi. Przedstawione w rozdziale drugim analizy wykazują, że precyzyjnych narzędzi do określenia takich kryteriów dostarczyć mogą semantyki języków programowania, a w szczególności – semantyki w pełni abstrakcyjne (zgodne oraz pełne). Pozwalają one bowiem na pominięcie szczegółów notacyjnych i implementacyjnych programów oraz spełniają prawo Leibniza – powszechnie przyjmowane kryterium służące do rozróżniania obiektów, będące połączeniem dwóch zasad: identyczności nierozróżnialnych oraz nieodróżnialności identycznych. Pojęciem spełniającym obie te zasady, które dobrze oddaje „równość" programów, jest ich obserwacyjna równoważność.

Semantyki języków programowania okazują się być użyteczne nie tylko w badaniu własności istniejących już programów, w tym ich równości, ale również podczas ich tworzenia. Pisanie programu jest złożonym procesem, rozpoczynającym się od sformułowania jego specyfikacji, a kończącym się sprawdzeniem poprawności. W rozdziale drugim poddano szczegółowej analizie trzy etapy tego procesu: tworzenie specyfikacji, implementację oraz weryfikację poprawności programów. Omówiono między innymi różne rodzaje specyfikacji – od sformułowanych w języku potocznym, poprzez formalne, aż po wykonywalne – analizując ich zalety i wady oraz związane z ich tworzeniem problemy filozoficzne. W jaki sposób odróżnić specyfikacje, szczególnie wykonywalne, od opisywanych przez nie programów? Czym jest specyfikacja? Czy jest ona definicją warunkową artefaktu? Czy może odpowiednikiem teorii naukowej? Przeanalizowano również wieloznaczny termin „implementacja", który jest rozumiany przez informatyków między innymi jako: przedstawienie algorytmu w języku programowania (program jest wtedy implementacją algorytmu), zapis kodu programu w języku maszynowym (w wyniku jego kompilacji lub interpretacji) oraz implementacja abstrakcyjnych typów danych w innych takich typach lub w konkretnych strukturach danych (na przykładzie implementacji stosu w postaci tablicy oraz listy). Czym zatem jest implementacja? Czy można podać definicję tego pojęcia, która obejmie wszystkie jego zastosowania w informatyce? Jedną z możliwości jest zdefiniowanie jej jako interpretacji semantycznej – relacji ternarnej pomiędzy dziedziną semantyczną, medium interpretacji a dziedziną syntaktyczną. Analiza wybranych przykładów implementacji pokazała, że nie wszystkie one są interpretacjami semantycznymi. A zatem rozumienie implementacji jako interpretacji semantycznej stanowi błędne utożsamienie dwóch istotnie różnych pojęć. Kwestia definiowania implementacji, nawet ograniczonego do informatyki, pozostaje nadal otwarta.

W rozdziale drugim omówiono również kwestie filozoficzne związane ze sprawdzaniem poprawności programów. Sam termin „poprawny" w odniesieniu do programu komputerowego można rozumieć dwojako: albo jako zgodny ze specyfikacją, albo też jako rozwiązujący stawiany przed nim problem. W jaki sposób możemy stwierdzić, w każdym z tych przypadków, że program jest poprawny? Próby odpowiedzi na to pytanie doprowadziły do powstania paradygmatu matematycznego w informatyce, którego zwolennicy i przeciwnicy od wielu lat toczą zaciekłe spory dotyczące dopuszczalnych metod tworzenia i sprawdzania programów komputerowych. Sympatycy tego paradygmatu koncentrują się na poprawności w pierwszym sensie, twierdząc, że konieczne jest tworzenie formalnych dowodów zgodności programów z ich specyfikacją; dopuszczają oni przy tym wykorzystanie specjalnych programów, tworzących tak zwane weryfikacje. Jego prze-

ciwnicy z kolei twierdzą, że nie jest możliwe matematyczne sprawdzenie poprawności programu, gdyż jest on jednym z elementów systemu przyczynowego (złożonego z urządzenia i oprogramowania), którego wiarygodność powinno się badać empiryczne – obserwując efekty jego działania. Do stwierdzenia poprawności programu – rozumianej jako działanie prowadzące do rozwiązania problemu – wystarczy zatem jego przetestowanie dla reprezentatywnego zbioru danych wejściowych.

Testowanie programów, widziane jako ich wykonanie w odpowiednich warunkach, służy nie tylko badaniu ich poprawności. Przeprowadzone analizy ukazały istotne związki pomiędzy różnymi rodzajami testowania. Można bowiem uruchomić program w celu zbadania: czy system komputerowy (rozumiany jako połączenie maszyny i wykonywanego na niej programu) działa poprawnie, czy program rozwiązuje stawiany przed nim problem bądź też w celu zweryfikowania pewnej hipotezy naukowej w komputerowym modelu rzeczywistości. W ostatnim przypadku programy (podobnie jak aparatura badawcza) są tylko narzędziami służącymi do odkrywania nieznanych dotąd praw natury, co prowadzi do wniosku, że programowanie – a nawet cała informatyka – jest nauką przyrodniczą.

W rozdziale trzecim omówiono pojęcie informacji, którą obecnie definiuje się najczęściej jako obiekt badań informatyki rozumianej jako nauka o jej przesyłaniu, przetwarzaniu, a nawet o jej naturze. Dokonano w nim podziału definicji terminu „informacja" na dwie grupy: ogólne – ujmujące ją w terminach wspólnych dla wszystkich nauk, oraz szczegółowe – określające informację w obrębie danej dyscypliny (czym innym jest wtedy informacja genetyczna, a jeszcze czym innym – prasowa). W literaturze pojawiają się liczne próby stworzenia ogólnej definicji informacji – ujmującej zarówno jej sens potoczny, jak i odpowiedniej dla wszystkich dziedzin nauki – na przykład w terminach efektów (wyjść) pewnych procesów, interpretowanych bardzo szeroko: od prostych funkcji matematycznych aż po złożone procesy społeczne. Takie ujęcia prowadzą zazwyczaj do zastąpienia niejasnego pojęcia „informacji" innymi – równie nieprecyzyjnym terminami – takimi jak „proces". Wydaje się zatem, że próby te skazane są na niepowodzenie. W przypadku określenia „informacja" mamy bowiem do czynienia nie tyle z pojedynczą koncepcją, ile raczej z całym labiryntem pojęć i związanych z nimi teorii – niektóre z nich przeanalizowano w rozdziale trzecim.

Informatycy rozumieją zazwyczaj informację w kategoriach czysto ilościowych, interesują ich bowiem metody jej mierzenia, przesyłania, przechowywania i wyszukiwania. Istnieje kilka ilościowych teorii informacji, jednak najszerzej znana i dyskutowana jest matematyczna teoria komunikacji Shannona. Nie jest to teoria informacji w sensie potocznym, gdyż rozważa się w niej jedynie informację zawartą w przesyłanych komunikatach.

Jednak przeważająca większość innych teorii opisujących informację przejmuje od Shannona model komunikacyjny oraz zasadę odwrotnej zależności stwierdzającą, iż więcej informacji niosą komunikaty bardziej zaskakujące (mniej prawdopodobne). Przyjęcie tej zasady związane jest z pewnym sprzecznym z intuicją stwierdzeniem – nazywanym paradoksem małp: całkowicie pozbawiony treści tekst (na przykład pisany przez małpy na maszynie do pisania) niesie więcej informacji niż dzieła Szekspira. Również każde z tych dzieł zawiera mniej informacji niż tekst złożony z losowo pomieszanych tworzących go symboli (liter i znaków przestankowych). Własność ta wynika ze sposobu mierzenia ilości informacji – największą jej ilość niesie bowiem taki tekst (ciąg znaków), w którym prawdopodobieństwo wystąpienia każdej litery jest takie samo, to znaczy ciąg całkowicie losowy.

Z paradoksem małp dobrze radzi sobie inna ilościowa teoria informacji – teoria złożoności algorytmicznej. Rozważa się w niej tak zwaną informację algorytmiczną dla danego ciągu jako długość najkrótszego programu (algorytmu), który go generuje – im krótszy jest ów program, tym mniejszą ilość informacji niesie ciąg. Takie ujęcie pozwala na obliczenie ilości informacji w przypadkach, dla których nie jest odpowiednia teoria Shannona (ze względu na oparcie jej na pojęciu prawdopodobieństwa), na przykład – informacji zawartych w twierdzeniach matematycznych, mapach czy zdjęciach.

Zarówno matematyczne teorie informacji (Shannona i teoria algorytmiczna), jak i oparta na terminach cybernetycznych teoria Mazura (opisana w paragrafie 3.2.3.) traktują informację jako wielkość fizyczną; takie ujęcie zazwyczaj odpowiada informatykom, ale nie zadowala filozofów, rozumiejących informację szerzej – nie tylko na poziomie syntaktycznym, ale również semantycznym. Szukają oni bowiem nie tylko sposobu mierzenia ilości informacji, ale również odpowiedzi na pytania związane z jej naturą: czy informacja może być fałszywa?, co czyni informację użyteczną?, jaki jest związek informacji z wiedzą?

Istnieją teorie informacji wykraczające w swych analizach poza jej aspekt czysto ilościowy, na przykład teorie informacji semantycznej, określające miarę treści semantycznej zawartej w komunikacie jako dopełnienie jego prawdopodobieństwa, interpretowanego w różny sposób: w terminach modalnych, w odniesieniu do obserwowanego stanu rzeczy, bądź też poprzez konstrukcję formuł atomowych w wybranym języku formalnym. Jednak niezależnie od rozumienia prawdopodobieństwa teorie semantyczne zachowują zasadę odwrotnej zależności, z której wynika paradoks małp. Nie jest to jedyna trudność z nimi związana, gdyż proponowane w nich rozumienie informacji prowadzi do paradoksu Bar-Hillela–Carnapa: zdania sprzeczne niosą najwięcej informacji. Sformułowanie wspomnianego paradoksu doprowadziło do wielu modyfikacji istniejących teorii informacji semantycznej oraz do powstania nowych. Jedną z nich jest teoria informacji silnie semantycznej,

definiująca informację jako zawartość semantyczną poprawnie sformułowanych, znaczących danych; jej ilość mierzona jest zgodnie z zasadą: zdanie niesie większą ilość informacji, gdy jest bardziej precyzyjne.

Jednym z najszerzej dyskutowanych zagadnień związanych z informacją semantyczną jest jej związek z prawdziwością. Czy informacja fałszywa to szczególny rodzaj informacji, czy też nie jest nią wcale? Jaki jest związek pomiędzy informacją, dezinformacją a informacją błędną?

Przeprowadzone analizy pokazują, że wykluczenie informacji fałszywej z zakresu pojęcia „informacja" prowadzi do znacznego jego zubożenia. Nie pozwala ono również na mówienie o informacji w przypadku zdjęć, map czy też plam krwi na miejscu zbrodni, gdyż nie sposób nazwać ich prawdziwymi. Podobne sytuacje dobrze opisuje teoria systemowa, zakładająca, że zawartość informacyjna zdania nie jest wyznaczona *a priori*, lecz w odniesieniu do danej sytuacji. Teoria pragmatyczna natomiast opisuje wpływ informacji, również fałszywej, na osiąganie pewnych celów; teoria semiotyczna z kolei wyjaśnia naturę informacji jako znaczących danych i nie ogranicza się tylko do informacji prawdziwej.

Przedstawione w rozdziale trzecim teorie informacji stanowią tylko część bardzo licznych prób opisu natury informacji. Badaniu tego pojęcia służy również filozofia informacji, która ma swoje źródła w cybernetyce, informatyce, sztucznej inteligencji i filozofii. Czym jest filozofia informacji? Można ją traktować jako dyscyplinę techniczną, której wyniki rzutują na wiele różnych dziedzin nauki, takich jak metodologia, epistemologia i etyka, bądź też jako zupełnie nową, która zrewolucjonizuje samą filozofię. Filozofia informacji w drugim znaczeniu stawia sobie dwa podstawowe cele: badanie natury i podstawowych zasad informacji (również jej dynamiki i zastosowań) – w tym między innymi, szukanie odpowiedzi na pytanie, czym jest informacja – oraz wprowadzenie nowej metodologii, służącej do rozwiązywania różnych kwestii filozoficznych. Jednakże nie ma ona być kolejną teorią informacji, podobną do przedstawionych powyżej, ale ma badać istniejące już teorie oraz pojęcia związane z informacją (istotne z punktu widzenia innych dyscyplin naukowych), takie jak: byt, wiedza, prawda, życie, znaczenie. Zdaniem niektórych badaczy, filozofia informacji, ze względu na bardzo bogaty aparat pojęciowy, którym dysponuje, może stać się teorią unifikującą wiele dyscyplin, wprowadzając wspólne dla nich ramy teoretyczne. Trudno jednak zgodzić się z poglądem, że jest to dojrzała dyscyplina naukowa, której wpływy obejmują prawie całą współczesną naukę. Nieuprawnione wydaje się również mówienie o „erze informacji" czy „nauce informacyjnej".

Większość trudności opisanych w rozdziale trzecim, a związanych z określeniem natury informacji, nie interesuje bezpośrednio informatyków, którzy koncentrują się głównie na sposobach jej reprezentowania, mierzenia, przechowywania, wyszukiwania i przesyłania. Jednak nawet

takie czysto ilościowe ujęcie informacji przyjmowane w informatyce wymaga refleksji filozoficznej. Czym jest informacja kwantowa? Jakie są filozoficzne konsekwencje odwracalności obliczeń kwantowych? W jaki sposób interpretować negację w bazach danych: czy właściwa jest logika klasyczna (z negacją rozumianą jako niepowodzenie), czy też któraś z logik wielowartościowych?

Reasumując, aby móc mówić (jak to się czyni powszechnie) o informacji jako o obiekcie badań informatyki, konieczne jest prowadzenie dalszych badań filozoficznych, mających na celu pełniejsze określenie natury zarówno samej informacji, jak i związanych z nią licznych pojęć i zjawisk.

W rozdziale czwartym przedstawiono zagadnienia ontologiczne związane z wirtualną rzeczywistością (wirtualnymi światami). Zjawisko to bowiem często rozumiane jest nie tylko jako wytwór informatyki, lecz także jako podstawowy przedmiot jej zainteresowań. Omówiono dwa podstawowe sposoby jej definiowania: w terminach psychologicznych – koncentrujący się na doznaniach podmiotu związanych z przebywaniem w niej – oraz za pomocą opisu technologii używanej do jej tworzenia. W przypadku pierwszym o rzeczywistości wirtualnej możemy mówić w całkowitym oderwaniu od informatyki, mamy z nią bowiem do czynienia na przykład podczas oglądania filmów, czytania książek czy też słuchania muzyki. Rzeczywistość wirtualna rozumiana jako tworzona przez komputery jest z kolei tworem zupełnie nowym, całkowicie zależnym od technologii. Nie dziwi więc fakt, że tak trudno jest podać jej cechy. W rozdziale czwartym poddano analizie szereg zjawisk związanych z rzeczywistością wirtualną, szukając tych, które ją definiują, między innymi: symulację, interaktywność, sztuczność, immersję, teleobecność, komunikację sieciową, fikcjonalizację, hipertekst oraz czas. Na podstawie tych analiz wysnuto wniosek, że wiele z nich to pojęcia wieloznaczne, a łączenie ich z rzeczywistością wirtualną prowadzi do różnego jej rozumienia. Nie można zatem w sposób jednoznaczny określić listy cech definiujących świat wirtualny, uwzględniającej dwa sposoby patrzenia na nią: psychologiczny i technologiczny.

Tworząc ontologię rzeczywistości wirtualnej, a w szczególności opisując sposób istnienia jej obiektów, wykorzystuje się najczęściej pojęcia „symulacji" oraz „fikcji". Obiekty wirtualne można traktować jako przybliżenia (symulacje) obiektów rzeczywistych, zarówno materialistycznie (jako wytwory komputera), jak i idealistycznie (jako niedoskonałe odbicia świata rzeczywistego). Można też widzieć obiekty świata wirtualnego jako powstające w procesie abstrakcji z obiektów rzeczywistych, czyli jako wytwór człowieka. Są one wówczas obiektami intencjonalnymi, istnieją, ponieważ człowiek chce obdarzyć je istnieniem. Nie wolno jednak w prosty sposób utożsamić szczególnego ich rodzaju – obiektów fikcyjnych – z wirtualnymi. Te drugie bowiem posiadają pewne cechy – takie jak dynamiczność, inter-

aktywność oraz ufundowanie na procesach cyfrowych – które nie charakteryzują tradycyjnych obiektów fikcyjnych.

Pierwszy człon wyrażenia „rzeczywistość wirtualna" sugeruje, że jest to pewnego rodzaju rzeczywistość. Czym zatem jest „rzeczywistość"? W podrozdziale 4.5 przeanalizowano koncepcję opisującą rzeczywistość jako fizyczną przestrzeń, w której mogą zachodzić ucieleśnione działania napotykające na opór, służące rozwiązywaniu problemów. Taka definicja pozwala na rozumienie świata wirtualnego jako pewnego rodzaju rzeczywistości. Powstaje zatem pytanie o związki łączące rzeczywistość wirtualną z realną.

Najczęściej traktuje się wirtualność jako pojęcie opozycyjne względem realności, nawiązując do klasycznego problemu filozoficznego rozróżnienia tego, co rzeczywiste, od tego, co pozorne. Jego rozwiązania poszukuje się wtedy przez porównanie obiektów wirtualnych do odbić lustrzanych bądź też doświadczeń związanych z przebywaniem w wirtualnej rzeczywistości – do marzeń sennych. Niektórzy badacze odrzucają jednak przeciwstawienie sobie pojęć realności i wirtualności, traktując rzeczywistość wirtualną jako pewne uzupełnienie realnego świata i podkreślając ich wzajemne przenikanie się w ludzkich doświadczeniach. Zależności pomiędzy tymi światami można rozważać w trzech aspektach: genetycznym (realność jest źródłem wirtualności), egzystencjalnym (istnienie wirtualności jest warunkiem istnienia wirtualności) oraz funkcjonalnym (podstawie funkcjonowania wirtualności – procedurom obliczeniowym – odpowiadają rzeczywiste mechanizmy, kierujące pracą komputerów).

Obiekty wirtualne można podzielić – uwzględniając ich związki z realnością – na dwa rodzaje: te, które są jedynie symulacjami obiektów rzeczywistych, oraz na te oddziałujące na rzeczywistość, prawdziwe reprodukcje ontologiczne – tak samo realne jak ich odpowiedniki w świecie rzeczywistym.

Na podstawie analiz zawartych w rozdziale czwartym dochodzimy do wniosku, że obiekty wirtualne mają pewien szczególny status ontologiczny. Nie istnieją fizycznie (nie mają masy i nie znajdują się w przestrzeni), ale nie są obiektami fikcyjnymi. Są natomiast interaktywne, gdyż zachodzi możliwość wzajemnych oddziaływań pomiędzy nimi a podmiotem. Co więcej, niektóre operacje na przedmiotach wirtualnych (przelewy bankowe czy handel) mają realny wpływ na naszą rzeczywistość, nie można zatem twierdzić, że są „nierealne". W związku z tym niektórzy badacze mówią o istnieniu pewnego rodzaju „przesunięcia ontologicznego" – choć świat wirtualny nie istnieje fizycznie, to istnieją jego realne skutki; wpływa on bowiem na człowieka podobnie jak rzeczywistość, zarówno na jego umysł, jak i na ciało.

Pojawienie się rzeczywistości wirtualnej rzuca też nowe światło na niektóre klasyczne problemy filozoficzne. Dobrym przykładem jest tu określenie związków pomiędzy umysłem a ciałem. W świecie rzeczywistości wirtualnej, nieposiadającej wymiaru fizycznego, potrzebne są pewnego rodzaju

wirtualne ciała (awatary), umożliwiające ludziom interakcję z tym światem; można traktować to jako argument na rzecz konieczności ucieleśnienia wszelkich form ludzkiego działania i poznania. W rzeczywistości wirtualnej nowy wymiar zyskała również tożsamość podmiotu – możliwe jest równoczesne istnienie wielu jego „ja". Ponadto łatwiejsze niż w realnym świecie staje się dokonywanie zmian w tożsamości – można ją bowiem edytować, ograniczać jej stosowanie, usuwać ją, a nawet tworzyć tożsamości fikcyjne. Pojęcie tożsamości (podobnie jak przestrzeni i czasu) zyskało zatem w wirtualnej rzeczywistości nowe cechy.

Zaprezentowane w niniejszej książce poglądy dotyczące definiowania informatyki jako nauki o *X* (komputerach, programach komputerowych, o informacji i rzeczywistości wirtualnej) można by poszerzyć o inne rozumienia obiektu jej badań. Niektórzy informatycy uważają na przykład, że centralnym pojęciem ich dyscypliny jest „algorytm", a inne obiekty – takie jak informacja, program czy komputer – pełnią jedynie funkcję pomocniczą. Programy są bowiem tylko implementacją algorytmów, a komputery umożliwiają ich wykonanie. W takim ujęciu algorytm jest wspólnym mianownikiem i pojęciem unifikującym wszystkie, coraz liczniejsze dyscypliny informatyki. Nawet w szkolnych podręcznikach znaleźć można definicje podobne do następującej: „Informatyka jest dziedziną wiedzy i działalności zajmującą się algorytmami" (Sysło, 1993, s. 16)[29]. Część badaczy kładzie nacisk na związki informatyki z szeroko rozumianą obliczalnością, twierdząc, że jest ona nauką o obliczaniu lub obliczalności. Abrahams (1987) proponuje nawet nową dla niej nazwę: „Moja osobista definicja dyscypliny i jej nazwa to 'computology': nauka o procesach obliczalnych i sposobach, w jaki mogą być one realizowane" (s. 473)[30].

W obliczu tak różnorodnych poglądów dotyczących obiektu badań informatyki oraz ze względu na liczne trudności związane z zaproponowanymi ich opisami trzeba szukać innych rozwiązań. Jednym z nich jest zdefiniowanie informatyki poprzez określenie metod, jakie stosują uprawiający tę naukę badacze. Jednak również w tym przypadku sytuacja nie jest prosta. Różne poglądy na temat ontologii informatyki znajdują swoje konsekwencje w sposobie patrzenia na metody pracy informatyków. Wśród ich aktywności wymienia się najczęściej: projektowanie, reprezentowanie i przetwarzanie informacji, programowanie, badania empiryczne, modelowanie. Również samą informatykę traktuje się wielorako: jako naukę przyrodniczą, matematykę, inżynierię i projektowanie, naukę ścisłą, sztukę, naukę społeczną

---

[29] Również David Harel (2001) zatytułował swoją książkę, powszechnie wykorzystywaną jako podręcznik akademicki, *Rzecz o istocie informatyki – algorytmika.*

[30] Również inni naukowcy proponowali dla informatyki nazwy podkreślające jej bliski związek z obliczaniem: „computics" (McKee, 1995) lub „computing science" (Dijkstra, 1989).

oraz jako rozważania interdyscyplinarne. Jakie jest zatem jej miejsce pośród innych nauk? Jakimi metodami powinno się ją uprawiać?

Współcześnie wyróżnić można cztery podstawowe paradygmaty informatyki, ściśle związane z określeniem jej jako nauki: matematyczny (nazywany też racjonalnym), zakładający, że informatyka jest gałęzią matematyki i tym samym właściwe do prowadzenia w niej badań są metody formalne; paradygmat technokratyczny – traktujący informatykę jako dziedzinę techniczną, dopuszczający wykorzystanie do jej uprawiania wszystkich metod inżynieryjnych, oraz paradygmat empiryczny (naukowy) – definiujący informatykę jako naukę przyrodniczą (ang. *natural science*), opartą na eksperymencie. Czwartym, dominującym obecnie poglądem jest traktowanie informatyki jako nowej dyscypliny wiedzy, która wykorzystuje zarówno osiągnięcia inżynierii, jak i metody formalne i eksperymentalne; nie jest ona jednak ani gałęzią matematyki, ani nauką przyrodniczą, ani też dziedziną inżynierii.

Trudno zatem określić, czym w istocie informatyka jest i jakie obiekty bada. Jak zdefiniować tę młodą dyscyplinę? Niektórzy informatycy uważają, że nazwa dla ich dyscypliny to tylko nic nieznacząca etykietka – „informatyka to przecież to, co robią informatycy". Jednak owa etykieta jest istotna z wielu powodów. Z jednej strony, ma ona duży wpływ na kwestie praktyczne: stanowi o statusie akademickim informatyki, oznacza pewien zawód. Wiele kwestii dotyczących danej dyscypliny zależy od tego, jak jest ona widziana z zewnątrz, zwłaszcza od sposobu, w jaki się ją określa. Z drugiej strony, samoświadomość i jasność konceptualna są ważną częścią każdej dojrzałej nauki. Co więcej, „[…] informatyka jako dyscyplina akademicka – lub to, o co w niej chodzi – musi być odróżnialna od innych rzeczy, takich jak robienie na drutach, podnoszenie podnośnikiem widłowym lub chemia; powinno istnieć jakieś wspólne pojmowanie tego, co rozumiemy przez informatykę" (Tedre, 2011, s. 383). Dick Hamming (1969, s. 4) podczas Wykładu Turinga[31] w roku 1968 ostrzegał:

> […] obraz, jaki ludzie mają dotyczący danej dziedziny, może w znaczący sposób wpływać na jej późniejszy rozwój. Dlatego też, chociaż nie możemy ostatecznie odpowiedzieć na pytanie [co to jest informatyka – przyp. I. B-K], to musimy co jakiś czas sprawdzać i wietrzyć nasze poglądy na to, czym jest i czym powinna się stać nasza dyscyplina.

---

[31] Wykłady Turinga (*The Turing Lecture*) wygłaszane są przez laureatów nagrody im. Alana Turinga (*A.M. Turing Award*) przyznawanej corocznie (od 1966 roku) przez *Association for Computing Machinery* (*ACM*).

Bibliografia

1. Abrahams, P. (1987). What Is Computer Science? *Communications of the ACM, 30*(6), 472-473.
2. Adriaans, P. (2013). Information. W: E.N. Zalta (red.), *The Stanford Encyclopedia of Philosophy* (Fall 2013 Edition). Pozyskano z: http://plato.stanford.edu/archives/ fall2013/ entries/information.
3. Adriaans, P., van Benthem, J. (red.) (2008). *Handbook of Philosophy of Information*. Amsterdam, Oxford: Elsevier.
4. Agresti, W.W. (1986). *New Paradigms in Software Development*. Washington: IEEE Computer Society Press.
5. Aho, A.V., Hopcroft, J.E. i Ullman, J.D. (1974). The Design and Analysis of Computer Algorithms. Boston: Addison-Wesley. Tłumaczenie polskie: *Projektowanie i analiza algorytmów komputerowych*. Warszawa: Helion 2003.
6. Angius, N., Tamburrini, G. (2011). Scientific theories of computational systems in model checking. *Minds and Machines*, *21*(2), 323-336.
7. Appel, K., Haken, W. (1983). Zagadnienie czterech barw. W: L.A. Steen (red.), *Matematyka współczesna. Dwanaście esejów* (s. 170-199). Warszawa: Wydawnictwa Naukowo-Techniczne.
8. Arora, S., Chazelle, B. (2005). Is the thrill gone? *Communications of the ACM*, *48* (8), 31-33.
9. Asperti A., Geuvers H. i Natarajan R. (2009). Social processes, program verification and all that. *Mathematical Structures in Computer Science*, *19*(5), 877-896.
10. Atchison W.F., Conte S.D. i in. (1968) Curriculum 68: Recommendations for Academic Programs in Computer Science: a Report of the ACM Curriculum Committee on Computer Science. *Communications of the ACM*, *11*(3), 151-197.
11. Aukstakalnis¸ S., Blatner¸ D. (1992). *Silicon Mirage; The Art and Science of Virtual Reality*. Berkeley, CA: Peachpit Press.
12. Baevers, A.F. (2011). Recent Developments in Computing and Philosophy. *Journal for the General Philosophy of Science*, *42*, 385-397.
13. Baker, L.R. (2000). *Persons and Bodies*. Cambridge University Press.
14. Baker, L.R. (2006). On the Twofold Nature of Artefact. *Studies in History and Philosophy of Science*, *37*, 132-136.

15. Balzer, R. (1985). A 15 Year Perspective on Automatic Programming. *IEEE Transactions on Software Engineering*, *11*, 1257-1267.
16. Banse, G. (2009). „Rzeczywistość wirtualna" i jej odniesienie do „rzeczywistości realnej". W: A.Kieps, M. Sułkowska i M. Wołek (red.), *Człowiek a światy wirtualne* (s. 42-49). Katowice: Wydawnictwo Uniwersytetu Śląskiego.
17. Bar-Hillel, Y., Carnap, R. (1952). *An Outline of a Theory of Semantic Information*, przedruk w: (Bar-Hillel, Y., 1964), 221-274.
18. Bar-Hillel, Y., Carnap, R. (1953). Semantic Information. *The British Journal for the Philosophy of Science*, *4* (14), 147-157.
19. Bar-Hillel, Y. (1964). *Language and Information*. London: Addison-Wesley.
20. Barwise, J. (1989). Mathematical Proofs of Computer System Correctness. *Notices of the American Mathematical Society 36*, 844-851.
21. Basili, V.R., Zelkowitz M.V. (2007). Empirical Studies. To Build a Science of Computer Science. *Communications of the ACM, 50*(11), 33-37.
22. Bateson, G. (1973). *Steps to an Ecology of Mind*. Frogmore, St. Albans: Paladin.
23. Baudrillard, J. (1983). Simulations. (przekład angielski: P. Foss. P. Patron, i P. Beitchman). New York: Semiotext(e).
24. Belnap, N. (1977). How a Computer Should Think. W: G. Ryle (red.), *Contemporary Aspects of Philosophy* (s. 30-55). Oriel Press.
25. Ben-Ari, M. (2006). *Understanding Programming Languages*. Chichester: John Wiley & Sons. Pozyskano z: http://pl.wikipedia.org/wiki/Program_komputerowy.
26. Berry, H., Perez, D.G. i Temam, O. (2006). Chaos in Computer Performance. *CHAOS*, *16*: 013110.
27. Bevier, W.R., Smith, M.K. i Young, W.D. (1989). Letter in the Technical Correspondence section of *Communications Computing of Association for Machinery*, *32*(3), 375-376.
28. Biorca, F. (1992). Communication Within Virtual Reality: Creating a Space for Research. *Journal of Communication*, *42*(4), 5-22.
29. Blum, B. (1989). Formalism and Prototyping in the Software Process. *Information and Decision Technologies*, *15*, 327-341
30. Bondecka-Krzykowska, I. (2010). Paradygmaty informatyki. W: I. Bondecka-Krzykowska, J. Pogonowski (red.), *Światy matematyki. Tworzenie czy odkrywanie?* (s. 117-134). Poznań: Wydawnictwo Naukowe UAM.
31. Bondecka-Krzykowska, I. (2012a). *Historia obliczeń. Od rachunku na palcach do maszyny analitycznej*. Poznań: Wydawnictwo Naukowe UAM.
32. Bondecka-Krzykowska, I. (2012b). Uwagi na temat ontologii wirtualnej rzeczywistości. *Filozofia Nauki*, *4*, 139-153.
33. Bornat, R. (2006). Is 'Computer Science' Science? W: *The 4th European Conference on Computing and Philosophy (ECAP'06)*, Trondheim: Norwegian University for Science and Technology.
34. Bostron, N. (2003). Czy żyjemy w Matriksie? Argument z symulacji. W: G. Yeffeth, *Wybierz czerwoną pigułkę. Nauka, filozofia i religia w Matrix* (s. 262-263). Gliwice: Helion.
35. Brey, P. (1998). Space-Shaping Technologies and the Geographical Disembedding of Place. W: A. Light, J. Smith (red.), *Philosophy & Geography Vol. III: Philospohies of Place* (s. 239-263). New York and London: Rowman Littlefield.

36. Brey, P. (2003). The Social Ontology of Virtual Environments. *American Journal of Economics and Sociology, 62*(1), 269-282.
37. Bringsjord, S. (1995). Computation, among Other Things, Is Beneath Us. *Minds and Machines*, *4*, 469-488.
38. Bromboszcz, R. (2004). Rozprawa z cyfrowym cieniem. O dwóch znaczeniach terminu „wirtualna rzeczywistość". *Estetyka i Krytyka*, *1*(6). Pozyskano z: http://estetykaikrytyka.pl/art/6/zzBromboszcz.pdf.
39. Brooks, F. (1996). The Computer Scientists as Toolsmith II. *Communications of the ACM*, *39*(3), 61-68.
40. Brożek, A. (2011). Filozofia nowej muzyki – rediviva. *Semina Scientiarum*, *10*, 10-20.
41. Burdea, G., Coiffet, P. (1994). *Virtual Reality Technology.* New York: John Wiley & Sons.
42. Bylina, J., Bylina, B. (2011). *Przegląd języków i paradygmatów programowania*. Lublin: Wydawnictwo Uniwersytetu Marii Curie-Skłodowskiej.
43. Campbell-Kelly, M., Aspray, W. (2004). *Computer: A History of the Information Machine* (2nd edition). Oxford UK: Westview Press.
44. Capurro, R. (2009). Past, Present, and Future of the Concept of Information. *Triple C 7* (2), 125-141.
45. Carnap, R. (1950). *Logical Foundations of Probability*. Chicago: University of Chicago Press.
46. Carnap, R. (1952). *The Continuum of Inductive Methods.* Chicago: University of Chicago Press.
47. Carnap, R. (1953). Formal and Factual Science. W: H. Feigl, M. Brodbeck (red.), *Readings in the Philosophy of Science* (s. 123-128). New York: Appleton-Century-Crofts.
48. Ceruzzi, P. (1989). Electronics Technology and Computer Science, 1940-1975: A Coevolution. *Annals of the History of Computing*, *10*(4), 257-275.
49. Chaitin, G.J. (1966). On the Length of Programs for Computing Finite Binary Sequences. *J. Association for Computing Machinery*, *13*(4), 547–569.
50. Chaitin, G.J. (1969). On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations. *Journal of the ACM*, *16*, 145-159.
51. Chaitin, G.J. (1977). Algorithmic Information Theory. *IBM Journal of Research and Development*, *21*, 350-359.
52. Chalmers, D.J. (1996). Does a Rock Implement Every Finite-State Automation? *Syntheses, 108*, 309-333.
53. Chesher, C. (1994). Colonizing Virtual Reality. Construction of the Discourse of Virtual Reality, 1984-1992. *Cultronix, 1*(1). Pozyskano z: http://cultronix.eserver. org/chesher.
54. Cleland, C.E. (2001). Recipes, Algorithms, and Programs. *Minds and Machines*, *11*, 219-237.
55. Cleland, C.E. (2002). On Effective Procedures. *Minds and Machines*, *12*(2), 159-79.
56. Clune, T.R. (1989). Letter in the Technical Correspondence section of *Communications of Association for Computing Machinery*, *32*(3), 375-376.
57. Cohen, M., Nagel, E. (1953). The Nature of a Logical or Mathematical System. W: H. Feigl, M. Brodbeck (red.). *Readings in the Philosophy of Science* (s. 129-147). New York: Appleton-Century-Crofts.

58. Cohen, J. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the Association for Computing Machinery*, *32*, 1408-1409.
59. Cohn, A. (1989). The Notion of Proof in Hardware Verification. *Journal of Automated Reasoning*, *5*(2), 127-139
60. Colburn, T.R. (1999). Software, Abstraction, And Ontology. *The Monist*, *82*(1), 3-19.
61. Colburn, T.R. (2000). *Philosophy and Computer Science*. M.E. Sharpe.
62. Colburn, T.R. (2004). Methodology of Computer Science. W: L. Floridi (red.) *The Blackwell Guide to the Philosophy of Computing and Information* (s. 318-326). Blackwell Publishing.
63. Colburn, T.R., Shute, G. (2011). Decoupling as Fundamental Value of Computer Science. *Minds and Machines*, *21*, 241-259.
64. Colburn, T., What is Philosophy of Computer Science? (extended abstract), maszynopis.
65. Copeland, B.J. (1996).What is Computation? *Synthese*, *108*, 335-359.
66. Copeland, B.J. (2002). Hypercomputation. *Minds and Machines*, *12*(4), 461-502.
67. Copeland, B.J., Shagrir, O. (2011). Do Accelerating Turing Machines Compute the Uncomputable? *Minds and Machines*, *21*(2), 221-239.
68. Crandal, R., Levich, M. (1998). *A Network Orange: Logic and Responsibility in the Computer Age*. New York: Copernicus.
69. Crowcroft, J. (2005). On the Nature of Computing. *Communications of the ACM*, *48*(2), 19-20.
70. Cummins, R. (1975). Functional Analysis. *The Journal of Philosophy*, *72*(20), 741-765.
71. DeMillo, R., Lipton, R. i Perlis, A. (1979). Social Processes and Proofs of Theorems and Programs. *Communications of ACM*, *22*, 271-280.
72. Denning, P.J. (1980). ACM President's Letter: On Folk Theorems, And Folk Myths. *Communications of the ACM*, *23*(9), 493-494.
73. Denning, P.J. (1981). ACM President's Letter: Performance Analysis: Experimental Computer Science As Its Best. *Communications of the ACM*, *24*(11), 725-727.
74. Denning, P.J. (1985). What Is Computer Science? *American Scientists*, *73*, 16-19.
75. Denning, P.J. (red.). (1989). A Debate On Teaching Computing Science. *Communications of the ACM*, *32* (12), 1397-1414.
76. Denning, P.J. (1995). Can There Be a Science of Information? *ACM Computing Surveys*, *27*(1), 23-25.
77. Denning, P.J. (1999). Computer Science: The Discipline. W: A. Ralston, E.D. Reily i D. Hemmendinger (red.), *Encyclopedia of Computer Science* (s. 2000-2034). Nature Pub. Group.
78. Denning, P.J. (2003). Great Principles of Computing. *Communications of the ACM*, *46*(11), 15-20.
79. Denning, P.J. (2005). Is Computer Science Science? *Communications of the ACM*, *48*(4), 27-31.
80. Denning, P.J. (2007) Computing Is a Natural Science. *Communications of the ACM*, *50*(7), 13-18.
81. Denning, P.J., Comer, D. i in. (1989). Computing as a Discipline. *Communications of the ACM*, *32* (1), 9-23.

82. Denning, P.J., Feigenbaum, E. i in. (1981). A Discipline in Crisis. *Communications of the ACM*, *24* (6), 370-374.
83. Denning, P.J., Freeman P.A. (2009). The Profession of IT. Computing's Paradigm. *Communications of the ACM*, *52*(12), 28-30.
84. Devlin, K. (1992). Computers and Mathematics. *Notices of the American Mathematical Society*, *39*, 1065-1066.
85. Dijkstra, E.W. (1972). The Humble Programmer. *Communications of the ACM*, *15*(10), 859-866.
86. Dijkstra, E.W. (1974). Programing as a Discipline of Mathematical Nature. *The American Mathematical Monthly*, *81*(6), 608-612.
87. Dijkstra, E.W. (1978). On a Political Pamphlet from Middle Ages. *ACM SIGSOFT Software Eng. Notes*, *3*(2), 14-16.
88. Dijkstra, E.W. (1986). On a Cultural Gap. *The Mathematical Intelligencer*, *8*(1), 48-52.
89. Dijkstra, E.W. (1989). On the Cruelty of Really Teaching Computing Science. *Communications of the ACM*, *32*, 1398-1404.
90. Dijkstra, E.W. (1997). The Tide, Not the Waves. W: P.J. Denning, R.M. Metcalfe (red.), *Beyond Calculation: The Next Fifty Years of Computing* (s. 59-64). New York: Springer.
91. Dipert, R.R. (1995). Some Issues in the Theory of Artifacts: Defining 'Artifact' and Related Notions. *The Monist*, *78*(2), 119-135.
92. Dobson, J., Randell, B. (1989). Program Verification: Public Image and Private Reality. *Communications of ACM*, *32*(4), 420-422.
93. Dodig-Crnkovic, G. (2006). What Is Philosophy of Computer Science? Experience from the Swedish National Course. Referat wygłoszony na konferencji ECAP-06.
94. Dodig-Crnkovic, G. (2011). Significance of Models of Computation, from Turing Model to Natural Computation. *Minds and Machines*, *21*(2), 301-322.
95. Dretske, F. (1981). *Knowledge and the Flaw of Information*. Cambridge: MIT Press.
96. Dretske, F. (1983). Precis of 'Knowledge and the Flaw of Information'. *Behavioral and Brain Sciences*, *6*, 55-90.
97. Duncan, W. (2009). Making Ontological Sense of Hardware and Software. Pozyskano z: http://www.cse.buffalo.edu/~rapaport/584/S10/duncan09-HWSWOnt.pdf.
98. Dunn, M. (2008). Information in Computer Science. W: D.M. Gabbay, P. Thagard i J. Woods (red.), *Philosophy of Information* (s. 589-616). Elsevier.
99. Easton, T.A. (2006). Beyond the Algorithmization of the Sciences. *Communications of the ACM*, *49*(5), 31-33.
100. Eden, A.H. (2001). Some Philosophical Issues in Computer Science. *Mind and Machines*, *21*, 123-133.
101. Eden, A.H., Kazman, R. (2003). Architecture, Design, Implementation. *Proc. 25th Int'l Confr. Software Engineering – ICSE*, Portland, IEEE Computer Society Press, 149-159.
102. Eden, A.H., Hirshfeld, Y. i Kazman, R. (2006). Abstraction Classes in Software Design, Technical report CSM-411. Department of Computer Science, University of Essex.
103. Eden, A.H., Turner, R. (2006). Problems in the Ontology of Computer Programs. Technical Report CSM-461, Department of Computer Science, University of Essex.

104. Eden, A.H. (2007). Three Paradigms of Computer Science. *Minds and Machines*, *17*, 135–167.
105. Egan, F. (1995). Computation and Content. *Philosophical Rewiev*, *104*, 181-204.
106. Emden, M.H. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the ACM*, *32*, 1407-1408.
107. Fant, K.M. (1993). A Critical Review of the Notion of Algorithm in Computer Science. *Proceedings of the 1993 ACM Annual Computer Science Conference.*
108. Feitelson, D.G. (2007). Special Issue on Experimental Computer Science. C*ommunications of the ACM*, *50*(11), 24-37.
109. Fetzer, J.H. (1988). Program Verification: The Very Idea. *Communications of Association for Computing Machinery*, *31*(9), 271-280.
110. Fetzer, J.H. (1989). Letter in the Technical Correspondence section of *Communications of Association for Computing Machinery*, *32*(3), 375-376.
111. Fetzer, J.H. (1991). Philosophical Aspects of Program Verification. *Minds and Machines*, *1*, 197-216.
112. Fetzer, J.H. (1999). The Role of Models in Computer Science. *The Monist*, *82*(1), 20-36.
113. Fetzer, J.H. (2004a). Information: Does it Have To Be True? *Minds and Machines*, *14*, 223-229.
114. Fetzer, J.H. (2004b). The Philosophy of AI and its Critique. W: L. Floridi (red.), *The Blackwell Guide to the Philosophy of Computing and Information* (s. 119-134). Oxford, New York: Blackwell Publishing.
115. Fletcher, P. (1995). The Role of Experiments in Computer Science. *Systems Software*, *3*, 161-163.
116. Floridi, L. (1999). *Philosophy and Computing. An Introduction*. London-New York: Routledge.
117. Floridi, L. (2002). What Is the Philosophy of Information? *Metaphilosophy*, *33*, 123-145.
118. Floridi, L. (2003). Two Approaches to Philosophy of Information. *Minds and Machines*, *13*, 459-469.
119. Floridi, L. (2004a). Outline of a Theory of Strongly Semantic Information. *Minds and Machines*, *14*, 197-221.
120. Floridi, L. (2004b). Open Problems in the Philosophy of Information. *Metaphilosophy*, *37*(5), 554-582.
121. Floridi, L. (2004c). Information. W: L. Floridi (red.), *The Blackwell Guide to the Philosophy of Computing and Information* (s. 40-61). Oxford, New York: Blackwell.
122. Floridi, L. (2005a). Information. W: C. Mitcham (red.), *Encyclopedia of Science Technology and Ethics*, Macmillan Reference.
123. Floridi, L. (2005b). Semantic Conceptions of Information. W: E.N. Zalta (red.), *Stanford Encyclopedia of Philosophy* (Spring 2015 Edition). Pozyskano z: http://plato.stanford.edu/archives/spr2015/entries/information-semantic.
124. Floridi, L. (2005c). Is Semantic Information Meaningful Data? *Philosophy and Philosophical Research*, *70*(2), 351-370.
125. Floridi, L. (2008). Trends in the Philosophy of Information, W: P. Adriaans, J. van Benthem (red.), *Handbook of Philosophy of Information* (s. 113-132). Amsterdam--Oxford: Elsevier.

126. Floridi, L. (2011). The Informational Nature of Personal Identity. *Minds and Machines*, *21*, 549-566.
127. Floyd, R. (1967). Assigning Meanings to Programs. *Proceedings of Symposia in Applied Mathematics*, *19*, 19-32.
128. Floyd, C. (1987). Outline of a Paradigm Change in Software Engineering. W: G. Bjerknes, P. Ehn i in. (red.), *Computers and Democracy: A Scandinavian Challenge* (s. 191-210). Hants, England: Gower Publishing Company.
129. Foley, J. (2002). Computing > Computer Science. *Computing Research News*, *14*(4), 6.
130. Forsythe, G.E. (1966). A University's Educational Program in Computer Science. *Technical Report no. CS39, Stanford University*.
131. Forsythe, G.E. (1967). A University's Educational Program in Computer Science. *Communications of the ACM*, *10*(1), 3-11.
132. Fox, C.J. (1983). *Information and Misinformation. An Investigation of the Notions of Information, Misinformation, Informing, and Misinforming*, Greenwood Publishing Group.
133. Freeman, P.A. (2008). Back to Experimentation. *Communications of ACM*, *51*(1), 21-22.
134. Fuchs, N.E. (1992). Specifications Are (Preferably) Executable. *Software Engineering Journal*, *7*(5), 323–334.
135. Gal-Ezer, J., Harel, D. (1998). What (Else) Should CS Educators Know? *Communications of the ACM*, *41*(9), 77-84.
136. Gettier, E.L. (1963). Is Justified True Belief Knowledge? *Analysis*, *23*, 121-123.
137. Grabiner, J.V. (1974). Is Mathematical Truth Time-Dependent, *Amer. Math. Monthly*, *81*, 354-365.
138. Gruner, S. (2011). Problems for a Philosophy of Software Engineering. *Minds and Machines*, *21*(2), 275-299.
139. Gumb, R.D. (1989). *Programming Logics: An Introduction to Verification and Semantics*. New York: John Wiley and Sons.
140. Goldin, D., Wegner, P. (2008). The Interactive Nature of Computing: Refuting the Strong Church-Turing Thesis. *Minds and Machines*, *18*, 17-38.
141. Gorn, S. (1963). The Computer and Information Sciences: a New Basic Discipline. *SIAM Review*, *5*(2), 150-155.
142. Grunwald, P.D., Vitanyi, P.M. (2008). Algorithmic Information Theory. W: P. Adriaans, J. van Benthem (red.), *Philosophy of Information* (s. 281-317). Amsterdam: North--Holland, Elsevier.
143. Gunn, J. (2003). Paradoks rzeczywistości w Matriksie. W: G. Yeffeth (red.). *Wybierz czerwoną pigułkę. Nauka, filozofia i religia w Matrix* (s. 67-78). Gliwice: Helion.
144. Gurczyński, J. (2004). Alexius Meinong i Roman Ingarden o intencjonalności i przedmiotach fikcyjnych. W: Z. Muszyński, J. Paśniczek (red.), *Intencjonalność jako kategoria filozofii umysłu i filozofii języka* (s. 67-84). Lublin: Wydawnictwo Uniwersytetu Marii Curie-Skłodowskiej.
145. Gurczyński, J. (2013). *Czym jest wirtualność. Matrix jako model rzeczywistości wirtualnej*. Lublin: Wydawnictwo Uniwersytetu Marii Curie-Skłodowskiej.
146. Güner, F. (2002). Virtuelle Realität. *Die Fontäne – Online, 18*. Pozyskano z: http:// diefontaene.de/archiv/nr-18/virt_realitaet01.htm.

147. Hall, A. (1990). Seven Myths of Formal Methods. *IEEE Software*, *7*(5), 11-19.
148. Hamming, R.W. (1969). One Man's View of Computer Science. *Journal of the ACM*, *16*(1), 3-12.
149. Hamming, R.W. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the Association for Computing Machinery*, *32*, 1409-1410.
150. Harel, D. (1980). On Folk Theorems. *Communications of the ACM*, *23*(7), 379-389.
151. Harel, D. (2001). *Rzecz o istocie informatyki. Algorytmika*. Warszawa: Wydawnictwo Naukowo-Techniczne.
152. Hartmanis, J. (1981). Nature of Computer Science and its Paradigms. *Communications of the ACM*, *24*(6), 353-354.
153. Hartmanis, J., Lin, H. (1992). What Is Computer Science And Engineering? W: J. Hartmanis, H. Lin (red.), *Computing the Future: A Broader Agenda for Computer Science and Engineering* (s. 163-216). Washington: National Academy Press.
154. Hartmanis, J. (1993). Some Observations about the Nature of Computer Science. W: R.K. Shyamasundar (red.), *Foundations of Software Technology and Theoretical Computer Science: 13th Conference, Bombay, India, December 15-17, Proceedings* (s. 1-12). Berlin: Springer-Verlag Lecture Notes in Computer Science 761.
155. Hartmanis, J. (1994). Turing Award Lecture on Computational Complexity and the Nature of Computer Science. *Communications of the ACM*, *37*(10), 37-43.
156. Hartmanis, J. (1995). On Computational Complexity and the Nature of Computer Science. *ACM Computing Surveys*, *27*(1), 7-15.
157. Hartmanis, J. (1995). Response to the Essays "On Computational Complexity and the Nature of Computer Science". *ACM Computing Surveys*, *27*(1), 59-61.
158. Hayes, P.J. (1988). *Computer Architecture and Organization*. McGraw Hill.
159. Hayes, P.J. (1997). What Is a Computer? An Electronic Discussion. *The Monist*, *80*(3), 389-404.
160. Hayes, I.J., Jones, C.B. (1990). Specifications Are Not (Necessarily) Harmful. *Software Engineering Journal*, *4*(6), 330-339.
161. Heidt, S. (1999). Floating, Flying, Falling. A Philosophical Investigation of Virtual Reality Technology. *Inquiry: Critical Thinking Across the Disciplines*, *18*(4), 77-98.
162. Heim, M. (1993). *The Metaphysics of Virtual Reality*. New York: Oxford University Press.
163. Heim, M. (1998). *Virtual Realism*. New York: Oxford University Press.
164. Hempel, C. (1953). On the Nature of Mathematical Truth. W: H. Feigl, M. Brodbeck (red.), *Readings in the Philosophy of Science* (s. 148-162). New York: Appleton-Century-Crofts.
165. Hetmański, M. (2013). *Epistemologia informacji.* Kraków: Copernicus Center Press.
166. Heyck, H. (2008a). Defining the Computer: Herbert Simon and Bureaucratic Mind – Part 1. *IEEE Annals of the History of Computing*, *30*(2), 42-51.
167. Heyck, H. (2008b). Defining the Computer: Herbert Simon and the Bureaucratic Mind – Part 2. *Annals of the History of Computing*, *30*(2), 52-63.
168. Hilpinen, R. (2011). Artifact. W: E.N. Zalta (red.), *The Stanford Encyclopedia of Philosophy*. Pozyskano z: http://plato.stanford.edu/archives/win2011/entries/artifact.
169. Hintikka, K.J.J. (1970). *On Semantic Information.* W: K.J.J. Hintikka, P. Suppes (red.), *Information and Inference* (s. 3-27). Dordrecht, Nederlands: Reidel.

170. Hoare, C.A.R. *(1969). An Axiomatic Basic for Computer Programming*. *Communications of the Association for Computing Machinery*, *12*(10), 576-580.
171. Hoare, C.A.R. (*1986). The Mathematics of Programming: an Inaugural Lecture Delivered before the Univ. of Oxford on Oct. 17, 1985. New York: Oxford University Press.*
172. Holloway, C.M. (1995). Software Engineering and Epistemology. *SIGSOFT Software Engineering Notes*, *20*(2), 20-21.
173. Hongladarom, S. (2011). Personal Identity and the Self in the Online and Offline World. *Minds and Machines*, *21*, 533-548.
174. Houkes, W., Meijers, A. (2006). The Ontology of Artefacts: the Hard Problem. *Studies in History and Philosophy of Science*, *37*(1), 118–131.
175. Houkes, W., Vermaas, P.E. (2004). Actions Versus Functions: A Plea for an Alternative Metaphysics of Artefacts. *The Monist*, *87*, 52-71.
176. Ifrah, G. (2006). *Historia powszechna cyfr*. Warszawa: Wydawnictwo W.A.B.
177. Ingarden, R. (1960-61). *Spór o istnienie świata*. Tomy I-III. Warszawa: Państwowe Wydawnictwo Naukowe.
178. Ingarden, R. (1988). *O dziele literackim. Badania z pogranicza ontologii, teorii języka i filozofii literatury*. Warszawa: Państwowe Wydawnictwo Naukowe.
179. Israel, D., Perry, J. (1990). What Is Information? W: P. Hanson (red.) *Information, Language and Cognition* (s. 1-15). Vancouver: University of British Columbia Press.
180. Israel, D., Perry, J. (1992). Information and Architecture W: J. Barwise, J.M. Gawron i in. (red.), *Proceedings of the Second Conference on Situation Theory and Its Applications Vol. 2*. (s. 147-160). Stanford, CA: Center for the Study of Language (CSLI).
181. Jadacki, J. (2003). Potoczne rozumienie „informacji". W: J. Jadacki (red.), *Analiza pojęcia informacji* (s. 7-9). Warszawa: Wydawnictwo Naukowe Semper.
182. Jones, C.B. (1980). *Software Developement: a Rigorous Approach*. New Jersey: Prentice-Hall, Englewood Cliffs.
183. Kant, I. (2001). *Krytyka czystego rozumu*. Kęty: Wydawnictwo Antyk.
184. Karp, R.M. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the ACM*, *32*, 1410-1412.
185. Kartezjusz (2010). *Medytacje o pierwszej filozofii.* Tom I. Seria: Wielcy Filozofowie. Warszawa: Wydawnictwo Naukowe PWN (pierwsze polskie wydanie 1958).
186. Khalil, H., Levy, L.S. (1978). The Academic Image of Computer Science. *ACM SIGCSE Bulletin*, *10*(2), 31-33.
187. Knuth, D.E. (1974). Computer Science and its Relation to Mathematics. *The American Mathematical Monthly*, *81*(4), 323-343.
188. Knuth, D.E. (1985). Algorithmic Thinking and Mathematical Thinking. *American Mathematical Monthly*, *92*, 170-181.
189. Knuth, D.E. (2001). *Things a Computer Scientists Rarely Talks About*. Stanford, CA: CSLI Publications.
190. Koba, G. (2009). *Informatyka. Podstawowe tematy. Podręcznik informatyki dla gimnazjum*. Wrocław-Warszawa: Wydawnictwo Szkolne PWN.
191. Kolmogorow, A.N. (1965). Three Approaches to the Quantitative Definition of Information. *Problems Inform. Transmission*, *1*(1), 1–7.

192. Konik, R. (2009). Wirtualność jako rehabilitacja iluzji. *Diametros*, *21*, 78-95.
193. Kroes, P. (2010). Engineering and the Dual Nature of Technical Artefacts. *Cambridge Journal of Economics*, *34*(1), 51-62.
194. Kroes, P. (2012). *Technical Artefacts: Creations of Mind and Matter: A Philosophy of Engineering Design*. Dordrecht: Springer.
195. Kroes, P., Meijers, A. (2006). The Dual Nature of Technical Artefacts. *Studies in History and Philosophy of Science*, *37*(1), 1-4.
196. Landin, P.J. (1964). The Mechanical Evaluation of Expressions. *The Computer Journal (British Computer Society)*, *6*(4), 308–320.
197. Latawiec, A. (2009). Rola symulacji w kreowaniu świata wirtualnego. W: A. Kieps, M. Sułkowska i M. Wołek (red.), *Człowiek a światy wirtualne* (s. 50-58). Katowice: Wydawnictwo Uniwersytetu Śląskiego.
198. Lenski, W. (2010). Information: A Conceptual Investigation. *Information*, *1*, 74-118.
199. Ligonniére, R. (1992). *Prehistoria i historia komputerów od początków rachowania do pierwszych kalkulatorów elektronicznych*. Kraków: Zakład Narodowy im. Ossolińskich.
200. Lloyd, J.W., Topor, R.W. (1984). Making Prolog More Expressive. *Journal of Logic Programming*, *1*(3), 225-240.
201. Loewenheim, U. (1989). Legal Protection for Computer Programs in West Germany. *Berkeley Technology Law Journal*, *4*(2), 187-215.
202. Losee, R.M. (1990). *The Science of Information: Measurement and Applications*. San Diego: Academic Press.
203. Losee, R.M. (1997). A Discipline Independent Definition of Information. *Journal of the American Society for Information Science*, *48*(3), 254-269.
204. Loui, M.C. (1995). Computer Science Is a New Engineering Discipline. ACM Computing Surveys (CSUR), 27(1), 31-32.
205. Lubański, M. (1975). *Filozoficzne zagadnienia teorii informacji*. Warszawa: Akademia Teologii Katolickiej.
206. Ługowska, D. (2009). Tożsamość a światy wirtualne. W: A. Kieps, M. Sułkowska i M. Wołek (red.), *Człowiek a światy wirtualne* (s. 108-115). Katowice: Wydawnictwo Uniwersytetu Śląskiego.
207. Mahoney, M.S. (2002). Software as Science – Science as Software. W: U. Hashagen, R. Keil-Slawik i A. Norberg (red.), *History of Computing: Software Issues* (s. 25-48). Berlin, Heidelberg, New York: Springer.
208. Markov, A.A. (1906). Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 2-ya seriya 15(94), 135-156 (również w: Markow, 1951).
209. Markov, A.A. (1951). *Izbrannie Trudy*. A.N.S.S.S.R., Leningrad.
210. Mazur, M. (1970). *Jakościowa teoria informacji*. Warszawa: Wydawnictwa Naukowo-Techniczne.
211. McCarthy, J. (1962). Towards a Mathematical Science of Computation. W: C.M. Popplewell (red.), *Proceedings of the IFIP Congress 62* (s. 21-28), North-Holland.
212. MacKenzie, D. (1992). Computers, Formal Proofs and the Law Courts. *Notices of the American Mathematical Society*, *39*, 1066-1069.

213. McKee, G. (1995). Computer Science or Simply 'Computics'? The Open Channel. *Computer*, *28*(12), 136.
214. McLaughlin, P. (2001). *What Functions Explain: Functional Explanation and Self-Reproducing Systems*. Cambridge: Cambridge University Press.
215. McLaughlin, B.P. (2004). Computationalism, Connectionism, and the Philosophy of Mind. W: L. Floridi (red.), *The Blackwell Guide to the Philosophy of Computing and Information* (s. 135-151). Oxford, New York: Blackwell Publishing.
216. Meijers, A.W.M. (2001). The Relational Ontology of Technical Artifacts. W: P.A. Kroes, A.W.M. Meijers (red.), *The Empirical Turn in the Philosophy of Technology* (s. 81-96). Amsterdam: Elsevier.
217. Meinong, A. (1960). On the Theory of Objects (tłumaczenie Über Gegenstandstheorie, 1904). W: R. Chisholm (red.), *Realism and the Background of Phenomenology* (s. 76-117). Glencoe, IL: Free Press.
218. Meyer, B. (1985). On Formalism in Specifications. *IEEE Software*, *2*(1), 6-26.
219. Miczka, T. (2009). Czysta, iluzja i testowanie realności: dwie rzeczywistości wirtualne – dwa uczestnictwa. W: A. Kieps, M. Sułkowska i M. Wołek (red.), *Człowiek a światy wirtualne* (s. 11-29). Katowice: Wydawnictwo Uniwersytetu Śląskiego.
220. Milne, R., Strachey, C. (1977). *A Theory of Programming Language Semantics*. New York: Halsted Press.
221. Minsky, M.L. (1970). Form and Content in Computer Science. *Communications of ACM*, *17*(2), 197-215.
222. Minsky, M.L. (1979). Computer Science and the Representation of Knowledge. W: M.L. Dertouzos, J. Moses (red.), *The Computer Age: A Twenty-Year View* (s. 392-421), Cambridge, MA and London, England: MIT Press.
223. Mooradian, N. (2006). Virtual Reality, Ontology, and Value. *Metaphilosophy*, *37*(5), 673-690.
224. Moore, J.H. (1978). Three Myths of Computer Science. *The British Journal for the Philosophy of Science*, *29*, 213-222.
225. Murawski, R. (1995). *Filozofia matematyki. Zarys dziejów*. Warszawa: Wydawnictwo Naukowe PWN.
226. Naps T.L., Nance, D.W. i Singh, B. (1989). *Introduction to Computer Science: Programming, Problem Solving, and Data Structures*. St. Paul, MN: West Publishing Company.
227. Naur, P. (1966). Proof of Algorithms by General Snapshots. *BIT Numerical Mathematics*, *6*(4), 310-316.
228. Naur, P. (1982). *Formalization in Program Development. BIT Numerical Mathematics*, *22*(4), 437-453.
229. Naur, P. (1992). The Place of Strictly Defined Notation in Human Insight. W: T.R. Colburn, J.H. Fetzer i T.L. Rankin (red.), P*rogram Verification: Fundamental Issues in Computer Science* (s. 261-274). Dordrecht, Holland: Kluwer Academic Publishers.
230. Naur, P. (2007). Computing Versus Human Thinking. *Communications of ACM*, *50*(1), 85-94.
231. Nawrocki, W. (2003). W poszukiwaniu istoty informacji. W: J. Jadacki (red.), *Analiza pojęcia informacji* (s. 37-62). Warszawa: Wydawnictwo Naukowe Semper.

232. Newell, A., Perlis, A.J. i Simon, H.A. (1967). Computer Science. *Science*, *157*, 1373-1374.
233. Newell, A., Simon, H.A. (1976). Computer Science as Empirical Inquiry. *Communications of the Association for Computing Machinery*, *19*(3), 113-126.
234. Parlante, N. (2005).What Is Computer Science? *Inroads – The SIGCSE Bulletin*, *37*(2), 24-25.
235. Parnas, D.L. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the Association for Computing Machinery*, *32*, 1405-1406.
236. Parnas, D.L. (1998). Software Engineering Programmes Are Not Computer Science Programmes. *Annals of Software Engineering*, *6*(1-4), 19-37.
237. Paulson, L., Cohen, A. i Gordon, M. (1989). Letter in the Technical Correspondence section of *Communications of Association for Computing Machinery*, *32*(3), 375-376.
238. Pierce, J.R. (1968). Keynote Address. Conference on Academic and Related Research programs in Computing Science (5-8 June 1967). W: A. Finerman (red.), *University Education in Computing Science*, New York: Academic Press.
239. Pincas, U. (2011). Program Verification and Functioning of Operative Computing Revisited: How about Mathematics Engineering? *Minds and Machines*, *21*(2), 337-359.
240. Piwek, Ł. (2008). The Dynamic Representation of Reality and of Ourself between Real and Virtual Worlds. W: A. Briggle, K. Waelbers i P. Brey (red.), *Current Issues in Computing and Philosophy* (s. 24-33). Amsterdam: IOS Press.
241. Plaice, J. (1995). Computer Science Is an Experimental Science. *ACM Computing Surveys*, *27*(1), 33.
242. Pleasant, J.C. (1989). Letter in the Technical Correspondence section of *Communications of Association for Computing Machinery*, *32*(3), 375-376.
243. Plotkin, G. (1981). *A Structural Approach to Operational Semantics*. Technical Report. DAIMI FN-19, Aarhus University.
244. Plotyn. (1959). *Enneady*. Tomy I-II, Warszawa: PWN.
245. Podsiad, A. (red.) (2000). *Słownik terminów i pojęć filozoficznych.* Warszawa: Instytut Wydawniczy PAX.
246. Pollock, J.L. (1987). Defeasible Reasoning. *Cognitive Science*, *11*, 481-518.
247. Pomorski, J.M. (1988). *Informacja i komunikacja. Pojęcia, wzajemne relacje*. Wrocław: Zakład Narodowy im. Ossolińskich.
248. Rajlich, V. (2006). Changing the Paradigm of Software Engineering. *Communications of the Association for Computing Machinery*, *49*, 67-70.
249. Rapaport, W.J. (1999). Implementation Is Semantic Interpretation. *The Monist*, *82*(1), 109-30.
250. Rapaport, W.J. (2005a). Philosophy of Computer Science: An Introductory Course. *Teaching Philosophy*, *28*(4), 319-341.
251. Rapaport, W.J. (2005b). Implementation Is Semantic Interpretation: Further Thoughts. *Journal of Experimental and Theoretical Artificial Intelligence, 17*(4), 385-417.
252. Reid, E. (1994). Cultural Formations in Text – Based Virtual Realities (niepublikowana praca magisterska). University of Melbourne.

253. Rheingold, H. (1991). *Virtual Reality*. New York: Summit Books.
254. Rosas, O. (2008). On the Ecological/Representational Structure of Virtual Environments. W: A. Briggle, K. Waelbers i P. Brey (red.), *Current Issues in Computing and Philosophy* (s. 13-23). Amsterdam: IOS Press.
255. Rosenbloom, P.S. (2004). A New Framework for Computer Science and Engineering. *Computer*, *37*(11), 23-28.
256. Salomonoff, R.J. (1960). *A Preliminary Report on a General Theory of Inductive Inference*, *Report V-131*. Cambridge, Ma.: Zator Co. (November Revision of February 4, 1960 report).
257. Sanders, A.F. (2006). Programs as Mathematical Objects. Abstract, konferencja ECAP 2006.
258. Savitzky, S. (1989). Letter in the Technical Correspondence section of *Communications of Association for Computing Machinery*, *32*(3), 375-376.
259. Saygin, A., Cicekli, I. i Akman, V. (2000). Turing Test: 50 Years Later. *Minds and Machines*, *10*, 463-518.
260. Scherlis, W.L., Scott, D.S. (1983). First Steps Towards Inferential Programming. *Information Processing, 83, 199-212.*
261. Scherlis, W.L. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the Association for Computing Machinery*, *32*, 1406-1407.
262. Searle, J.R. (1980). Minds, Brains, and Programs. *Behavioral and Brain Sciences*, *3*, 417-457.
263. Searle, J.R. (1990). Is the Brain a Digital Computer? *Proceedings and Addresses of the American Philosophical Association*, *64*, 21-37.
264. Searle, J.R. (1998). *The Construction of Social Reality*. Cambridge: MIT Press.
265. Sequoiah-Grayson, S. (2007). The Metaphilosophy of Information. *Minds and Machines 17*, 331-344.
266. Searle, J.R. (1995). *The Construction of Social Reality*. London: Penguin.
267. Shagrir, O. (1999). What is Computer Science About? *The Monist*, *82*, 131-149.
268. Shannon, C.E., Waver, W. (1948). The Mathematical Theory of Communication. *Bell System Technical Journal, 27*(3), 379–423.
269. Shannon, C.E., Waver W. (1998). *The Mathematical Theory of Communication*. Urbana: University of Illinois Press [pierwsze wydanie 1948].
270. Shannon, C. (1953). The Lattice Theory of Information. *Transactions of the IRE Professional Group on Information Theory (TIT)*, *1*(1), 105-107.
271. Shapiro, S. (1994). Reasoning, Logic and Computation. *Philosophia Mathematica*, *3*(2), 31-51.
272. Shapiro, S. (2001). Computer Science: The Study of Procedures. Maszynopis.
273. Shneiderman, B. (2007). Web Science: A Provocative Invitation to Computer Science. *Communications of the ACM*, *50*(6), 25-27.
274. Simon, H.A. (1981). *The Sciences of Artificial.* Cambridge: MIT Press.
275. Sitarski, P. (2002). *Rozmowa z cyfrowym cieniem. Model komunikacyjny rzeczywistości wirtualnej*. Kraków: RABID.
276. Smith, B.C. (1985). Limits of Correctness in Computers, Report No. CSLI-85-36. Center for Study of Language and Information.

277. Smith, B. (2004). Ontology. W: L. Floridi (red.), *The Blackwell Guide to the Philosophy of Computing and Information* (s. 155-166). Oxford, New York: Blackwell Publishing.
278. Spence, E.H. (2008). Meta Ethics for the Metaverse: The Ethics of Virtual Worlds. W: A. Briggle, K. Waelbers i P. Brey (red.), *Current Issues in Computing and Philosophy* (s. 3-12). Amsterdam: IOS Press.
279. Stanovsky, D. (2004). Virtual Reality. W: L. Floridi (red.), *The Blackwell Guide to the Philosophy of Computing and Information* (s. 167-177). Oxford, New York: Blackwell Publishing.
280. Steuer, J. (1992). Defining Virtual Reality: Dimensions Determining Telepresence. *Journal of Communication*, *42*(4), 73-93.
281. Suber, P. (1998). What Is Software? *Journal of Speculative Philosophy*, *2*, 89-119.
282. Sysło, M. (red.). (1993). *Elementy informatyki. Podręcznik*. Warszawa: Wydawnictwo Naukowe PWN.
283. Talbott, S.L. (1999). There Is No Such Thing as Information, NETFUTURE 84. Pozyskano z: http://www.netfuture.org/1999/Feb0999_84.html.
284. Tedre, M. (2007). Lecture Notes in The Philosophy of Computer Science. Maszynopis.
285. Tedre, M. (2007). Know Your Discipline: Teaching the Philosophy of Computer Science. *Journal of Information Technology Education*, *6*, 105-121.
286. Tedre, M. (2009). Computing as Engineering. *Journal of Universal Computer Science*, *15*(8), 1642-1658.
287. Tedre, M. (2011). Computing as a Science: A Survey of Competing Viewpoints. *Minds and Machines*, *21*, 361-387.
288. Tedre, M. (2015). *The Science of Computing: Shaping a Discipline*. CRC Press.
289. Thomasson, A. (2007). Artifacts and Human Concepts. W: S. Laurence, E. Margolis (red.), *Creations of the Mind: Essays on Artifacts and Their* Representations (s. 52-73). Oxford: Oxford University Press.
290. Thurston, W.P. (1994). On Proof and Progress in Mathematics. *Bulletin of American Mathematical Society*, *30*, 161-171.
291. Tichy, W.F., Lukowicz, P. i in. (1995). Experimental Evaluation in Computer Science: A Quantitive Study. *Journal of Systems and Software*, *28*(1), 9-18.
292. Tichy, W.F. (1998). Should Computer Scientists Experiment More? *Computer*, *31*(5), 32-40.
293. Tucker, A.B., Wegner, P. (1997). Computer Science and Engineering: The Discipline and Its Impact. W: Tucker, Jr., Allen B. (red.), *The Computer Science and Engineering Handbook* (s. 1-15). Boca Raton, FL, USA: CRC Press.
294. Turing, A. (1936). On Computable Numbers with an Application to the Eintscheidungsproblem. *Proceedings of the London Mathematical Society*, *42*(2), 230-265.
295. Turing, A.M. (1950). Computing Machinery and Intelligence. *Mind*, *59*, 433-460.
296. Turner, R. (2005). The Foundations of Specification. *Journal of Logic and Computation*, *15*(5), 623-663.
297. Turner, R. (2007). Understanding Programming Languages. *Minds and Machines*, *17*(2), 203-216.
298. Turner, R. (2011). Specification. *Minds and Machines*, *21*, 135-152.

299. Turner, R. (2013a). The Philosophy of Computer Science. W: E.N. Zalta (red.), *The Stanford Encyclopedia of Philosophy*. Pozyskano z: http://plato.stanford.edu/entries/computer-science.
300. Turner, R. (2013b). Programming Languages as Technical Artefacts. *Philosophy and Technology*. DOI: I 10.1007/s13347-012-0098-z.
301. Turner, R., Eden, A.H. (2007). Towards a Programming Language Ontology. W: G. Dodig-Crnkovic (red.), *Computation, Information, Cognition: The Nexus and the Liminal* (s. 147-159). Cambridge: Cambridge Scholar Press.
302. Turner, R., Eden, A.H. (2011). The Philosophy of Computer Science. W: *Stanford Encyclopedia of Philosophy* (Fall 2013 Edition). Pozyskano z: http://plato.stanford.edu/ archives/fall2011/entries/computer-science.
303. Vaccari, A. (2013). Artifact Dualism, Materiality, and the Hard Problem of Ontology: Some Critical Remarks on the Dual Nature of Technical Artifacts Program. *Philosophy & Technology*, *26*(1), 7-29.
304. Vermaas, P.E., Houkes, W. (2003). Ascribing Functions to Technical Artifacts: A Challenge to Etiological Accounts of Function. *British Journal of the Philosophy of Science*, *54*, 261-289.
305. Weaver, W. (1949). The Mathematics of Communication. *Scientific American*, *181*(1), 11-15.
306. Wegner, P. (1970). Three Computer Cultures – Computer Technology, Computer Mathematics and Computer Science. *Advances in Computers*, *10*, 7-78.
307. Wegner, P. (1976). Research Paradigms in Computer Science. W: Proceedings of the 2nd International Conference on Software Engineering, *San Francisco, CA* (s. 322-330).
308. Wegner, P. (1999). Towards Empirical Computer Science. *The Monist*, *82*(1), 58-108.
309. Weizenbaum, J. (1976). *Computer Power and Human Reason: From Judgment to Calculation*. San Francisco: W.H. Freeman.
310. Weizsäcker, C.F., von. (1978). *Jedność przyrody*. Warszawa: PIW.
311. Wheeler, J. (1990). Information, Physics, Quantum: The Search for Links. W: W.H. Zurek (red.), *Complexity, Entropy, and the Physics of Information* (s. 309-336). Redwood: Addison-Wesley.
312. White, G. (2004). The Philosophy of Computer Languages. W: L. Floridi (red.), *The Blackwell Guide to the Philosophy of Computing and Information* (s. 237-247). Oxford, New York: Blackwell Publishing.
313. White, G. (2011). Descartes Among the Robots: Computer Science and the Inner/Outer Distinction. *Minds and Machines*, *21*(2), 179-202.
314. Wiener, N. (1954). *The Human Use of Human Beings: Cybernetics and Society*. Boston: Houghton Mifflin.
315. Wiener, N. (1961). *Cybernetics or Control and Communication in the Animal and the Machine*. Cambridge, Mass.: MIT Press.
316. Wilkoszewska, K. (2005). Wprowadzenie. W: M. Ostrowicki (red.), *Estetyka wirtualności* (s. 8). Kraków: Universitas.

317. Wing, J.M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33-35.
318. Wing, J.M. (2008). Five Deep Questions in Computing. *Communications of the ACM*, *50*(1), 58-60.
319. Winograd, T. (1989). Colleagues Respond to Dijkstra's Comments. *Communications of the Association for Computing Machinery*, *32*, 1412-1413.
320. Winograd, T. (1997). The Design of Interaction. W: P.J. Denning, R.M. Metcalfe (red.), *Beyond Calculation: The Next Fifty Years of Computers* (s. 149-161). New York: Copernicus.
321. Wulf, W.A. (1995). Are We Scientists or Engineers? *ACM Computing Surveys*, *27*(1), 55-57.

From ontological issues of computer science

S u m m a r y

Computer science is a relatively new scientific discipline with a distinctive place in our world and whose products are omnipresent. It is difficult to imagine a world without computers or the Internet. What is more, computer systems are being used with increasing frequency in many scientific fields, not only to perform complicated calculations, but also for experiments and to prove mathematical propositions.

This book takes a look at computer science through the eyes of the philosopher. In seeking an answer to the question 'what is computer science?' and its status among the sciences, an attempt is made to define computer science through defining the subjects it researches. For this reason the book also contains analysis of selected philosophical questions related to computers, computer programmes, information and virtual reality.

A computer scientist is often employed as a specialist in computer devices. However, the computer can be understood in two ways: as a device (or part of it) and as an abstract object. Yet creating the ontology of computers is no easy task, and the book presents the possibilities connected with using existing ontologies of technical artefacts for this purpose, as well as the difficulties related to doing so, in particular questions related to defining the function of artefacts and presenting criteria for their correct operation. The analysis also focuses on the consequences of defining computers in purely synthetic terms – as mathematical objects – including the issue of universal realizability. Furthermore, the relations between the terms 'hardware' and 'software' are investigated, aloong with an attempt to define them as separate, unequivocably classified entities.

Many computer scientists maintain that their primary task is to create programmes and research their properties. However, the term 'programme' itself appears in different contexts in this subject. In the book a distinction is made between programmes as abstract objects and as a physical process (performed on the computer). Different philosophical aspects related to the process of creating computer programes are also analysed: from formulating specifications through to checking their accuracy. It is shown how the dual (simultaneously abstract and physical) nature of programmes presents classic philosophical problems in a new light, such as the relation between the mind and body, and the division into abstract and concrete entities. Programmes are sometimes treated as objects with a specific ontological status and which are neither concrete nor

abstract. Their ontology may be sought, however, in the philosophy of music or of mathematics.

At present, the dominant view is that computer science is the science of information transformation processes or of information itself. The book therefore makes an attempt to answer questions about the nature of information, for this term is used both in colloquial speech and in many fields of science, ranging from information technology to to the social sciences. However, what information is to the computer scientist and what genetics is to the biologist are different things, and this is why this book discusses the philosophical aspects of selected theories of information, placing particular emphasis on mathematical theories used in computer science.

Computer systems are more and more frequently used as a tool for creating what is known as virtual reality (virtual worlds). This book analyses attempts to define this reality as well as the characteristic qualities of phenomena defined as such. It also describes the onotological status of virtual objects, the relations between the virtual and the real (the real world) and changes in understanding certain classical metaphysical concepts emerging along with virtual worlds.

Translated by Rob Pagget