ROCZNIKI FILOZOFICZNE
Tom LVIII, numer 2 – 2010

KAZIMIERZ TRZĘSICKI

LOGIKA TEMPORALNA W INFORMATYCE

1. NA POCZĄTKU BYŁA LOGIKA

Logika ma podstawowe znaczenie dla informatyki zarówno dla jej problemów teoretycznych, jak i technologicznych, zarówno jeśli chodzi o programy, jak i jeśli chodzi o sprzęt. Miejsce logiki w informatyce skrótowo można opisać stwierdzeniem: k o m p u t e r j e s t w y t w o r e m l o g i k i i t e c h n o l o g i i. Problematyce logicznej w informatyce poświęcone są liczne wyspecjalizowane czasopisma, jak np. amerykańskie „Logical Methods in Computer Science”[1] i „ACM Transactions on Computational Logic”[2], konferencje, jak np. *IEEE Symposium on Logic in Computer Science*[3]. Oczywiście, nie brakuje różnego rodzaju publikacji zbiorowych i monografii. W związku z tym, że w programach nauczania informatyków przewidziane jest nauczanie logiki (zwykle łącznie z teorią mnogości) nie brakuje odpowiednich podręczników.

Kwestie logiczne występują w problematyce[4]:

– modelowania systemów komputerowych i rozumienia ich architektury,
– języków bazodanowych,
– języków programowania (semantyka, logika programowania),
– sztucznej inteligencji jako narzędzia reprezentacji i rozumowania,
– inżynierii programowania (specyfikacja, weryfikacja i optymalizacja programów i systemów komputerowych),
– teorii obliczalności (pojęcie złożoności).

Prof. dr hab. KAZIMIERZ TRZĘSICKI – Katedra Logiki, Informatyki i Filozofii Nauki, Uniwersytet w Białymstoku; adres do korespondencji: ul. Sosnowa 64, 15-887 Białystok, pokój 205; email: kasimir@uwb.edu.pl

[1] http://www.lmcs-online.org/index.php
[2] http://tocl.acm.org
[3] http://www2.informatik.hu-berlin.de/lics/index.html
[4] Zob. www.cs.rice.edu/vardi/comp409

Logika jest jedną z najstarszych nauk. Jej początki wiążą się ze starożytną filozofią grecką, a Arystoteles powszechnie uważany jest za jej twórcę. Nadał on logice tak doskonały kształt, że jeszcze w XVIII wieku Immanuel Kant (1724-1804) uważał, że prawie niczego już do niej nie można dodać. Pisał on, że logika[5]:

> od czasów Arystotelesa nie musiała zrobić żadnego kroku wstecz […]. Osobliwe jest jeszcze to, że nie mogła dotychczas zrobić także ani kroku naprzód i że przeto wedle wszelkich danych wydaje się zamknięta i wykończona.

Elektronika rozwinęła się zaś w połowie wieku XX. Dopiero jej rozwój umożliwił zbudowanie odpowiedniego sprzętu.

Liczenie jest trudną operacją myślową, a wynik liczenia powinien być dokładny. Ludzie, od kiedy zaczęli liczyć, wspierali ten proces myślowy za pomocą różnego rodzaju narzędzi zarówno w zakresie operacyjnym, jak i zapamiętywania. Rajmundusa Lullusa wskazuje się jako kogoś, kto miał pewne intuicje możliwości tego rodzaju wspierania innych niż tylko obliczanie procesów myślowych. Idea taka została wyraźnie sformułowana przez Leibniza.

Odkrycie zera i stworzenie systemu pozycyjnego oraz wynalazek algorytmu pozwoliły zastąpić liczenie przez algorytmiczne rachowanie[6]. Dzięki temu niepomiernie wzrosły możliwości liczenia na bardzo dużych liczbach, a ponadto osiągnięto niemożliwą wcześniej bezbłędność. Rachowanie można zaś powierzyć maszynie. Mówi o tym Leibniz [26]:

> Indignum est excellentium virorum horas servili calculandi labore perire quia

[5] I. K a n t, *Krytyka czystego rozumu*, tł. R. Ingarden, Warszawa 1986, B VIII (s. 21-22).

[6] Jednym z pierwszych spośród tych, którzy używali i popularyzowali system dziesiętny, był Gerbert z Aurillac (ok. 955-1003), najwybitniejszy matematyk chrześcijańskiej średniowiecznej Europy (wybitniejsi matematycy byli w krajach islamskich). Studia w zakresie *quadrivium* odbył w Barcelonie. Katalonia utrzymywała kontakty z muzułmańską Al-Andalus. Dzięki temu Gerbert miał okazję zapoznać się z dorobkiem uczonych arabskich z Kordoby. Spośród wielu osiągnięć Gerberta w kontekście naszych rozważań warto wspomnieć o abaku, który jest praprzodkiem komputera. Gerbert wykorzystał posadzkę katedry w Reims. Sześćdziesięciu czterech uczniów szkoły katedralnej przestawiało kręgi. W ten sposób dawał sobie radę z wielkimi liczbami, z jakimi nigdy przedtem sobie nie radzono. Jest autorem rozpraw *Regula de abaco computi* („Reguły rachowania na abaku") i *Libellus de numerorum divisione* („Książeczka o dzieleniu liczb"). Doprowadził dzięki temu do wzrostu zainteresowania i zrewolucjonizowania studiów matematycznych na Zachodzie. Umiejętność obliczania wyników operacji na wielkich liczbach tak zadziwiała, że zarzucano mu paktowanie z diabłem. Z protekcji cesarza Gerbert został papieżem i przyjął imię Sylwestra II (lata pontyfikatu: 999-1003). Razem z cesarzem jako obcy zostali wygnani z Rzymu.

> Machina adhibita velissimo cuique secure transcribi possit. (Nie jest godne, aby znakomici ludzie tracili czas jak niewolnicy, wykonując obliczenia, gdy te mogą być powierzone komukolwiek, kto z pomocą maszyny poprawnie je przeprowadzi).

Leibniz skonstruował maszynę liczącą. Na cześć swojej maszyny zaprojektował medal z napisem: „Temu, co przewyższa człowieka".

Niezawodność rachowania oraz obiektywna sprawdzalność jego poprawności mogła być wzorem dla dysput w innych obszarach rozumowań. Ta idea owładnęła Leibnizem. Uznał, że dla tego celu potrzebny jest specjalny język pojęciowy, *characteristica universalis*, język Adama, oraz specjalny rachunek *calculus raciocinator*. Zdaniem Davisa [13] te spekulacje Leibniza są wizją Uniwersalnej Maszyny Turinga. Oparciem dla takiego celu była koncepcja metafizyczna. Świat został stworzony według liczby, miary i wagi (Mdr 11, 20). Leibniz pisze: „Cum Deus calculat et cogitationem excercet, fit mundus" („Kiedy Bóg liczy i snuje myśli, powstaje świat")[7].

Matematyka jest narzędziem Konstruktora świata, a tworzywem, z którego świat jest stworzony, są liczby. Idea Boga jako matematyka przez nikogo wcześniej nie była podkreślana tak silnie, jak uczynił to Leibniz. Stworzony przez siebie system binarny uznał za ,,obraz stworzenia" (*imago creationis*)[8]. Na zaprojektowanym przez siebie medalu na cześć systemu binarnego umieścił m.in. napis:

Unus ex nihilo omnia
Unum autem necessarium[9].

W styczniu 1697 r. wraz z życzeniami urodzinowymi do swego protektora księcia Rudolfa Augusta z Brunszwika (Herzog von Braunschweig-Wolfenbüttel Rudolph August) przesłał list, w którym omawia system binarny i ideę stworzenia z zera (0) jako nicości i jedynki (1) jako oznaczającej Boga. Dla Leibniza [27]:

> Ponieważ jedną z głównych kwestii wiary chrześcijańskiej, a mianowicie wśród tych, w które mądrości świata najmniej weszły oraz jeszcze poganom nie w pełni są ukazane, jest stworzenie rzeczy z niczego przez Wszechmocnego Boga. Teraz można właściwie powiedzieć, że nic w świecie tego zarówno lepiej nie przed-

[7] Zdanie to zamieszczone jest na marginesie tekstu *Dialogus*, opublikowanego w VII tomie pism G.W. Leibniza pt. *Die Philosophischen Schriften von G. W. Leibniz*, vol. I-VII, wydanych przez C.I. Gerhardta (Halle 1846-1863; reprint Hildesheim 1960), na s. 190-193.

[8] Idea ta została rozwinięta przez Chaitina. Więcej na ten temat zob. [41].

[9] „Jedynka z nicości (stwarza) wszystko. Jedynka zatem konieczna".

stawia, jak i jednocześnie dowodzi, jak powstanie liczb, jak tutaj jest przedstawione, przez wyrażenie ich tylko i zaledwie jedynką i zerem lub nicością, z czego wszystkie liczby powstają. I będzie w pełni trudno w przyrodzie i filozofii znaleźć lepsze przestawienie tajemnic, stąd także naszkicowany medal przedkładam:

OBRAZ STWORZENIA

To jest także warte nie mniejszej uwagi, jak już z tego ukazuje się, nie tylko, że Bóg wszystko uczynił z niczego, lecz także, że Bóg wszystko dobrze uczynił, i że wszystko, co uczynił, było dobre; jak to także tu w tym przedstawieniu stworzenia na oczy widzimy[10].

Leibniz projektuje również komputer binarny (mechaniczny).

Leibniz, podobnie jak wcześniej Thomas Hobbes, który głosił, że „c o g i t a t i o e s t c o m p u t a t i o", żywił przekonanie, że każde ludzkie rozumowanie może być przekształcone tak, że stanie się przedmiotem rachunkowego obliczenia i w taki sposób każda kontrowersyjna prawda stanie się tak oczywista jak 2 + 2 = 4. Pisał [28: 200]:

…gdyby spór powstał, dysputa między dwoma filozofami nie wymagałaby większego wysiłku niż między dwoma rachmistrzami. Wystarczyłoby bowiem, aby wzięli ołówki w swoje ręce, usiedli przy swoich tabliczkach i jeden drugiemu (z przyjacielem jako świadkiem, gdyby zechcieli) powiedzieli: P o l i c z m y[11].

W kwietniu 1679 r. w liście do księcia Johanna Friedricha von Braunschweig pisał:

[10] „Denn einer der Hauptpuncten des christlichen Glaubens, und zwar unter denjenigen, die den Weltweisen am wenigsten eingegangen, und noch den Heyden nicht wohl beizubringen sind, ist die Erschaffung der Dinge aus Nichts durch die Allmacht Gottes. Nun kann man wohl sagen, daß nichts in der Welt soie besser vorstelle, ja gleichsam demonstrire, als der Ursprung der Zahlen, wie er allhier vorgestellet ist, durch deren Ausdrückung blos und allein mit Eins und mit Nulle oder Nichts alle Zahlen entstehen. Und wird wohl schwerlich in der Natur und Philosophie ein bessres Vorbilddieses Geheimnisses zu finden sein, daher ich auch die entworfene Medaille gesetzet: *IMAGO CREATIONIS*. Es ist aber doch dabei nicht weniger betrachtungswürdig, wie schon darus erscheinet, nicht nur, daß Gott Alles aus Nichts gemacht, sondern auch daß Gott Alles wohl gemacht, und daß Alles, was er geschaffen, gut gewesen; wie wirs hier denn in diesem Vorbilde der Schöpfung auch mit Augen sehen".

[11] „… quando orientur controversiae, non magis disputatione opus erit inter duos philosophos, quam inter duos Computistas. Sufficiet enim calamos in manus sumere sedereque ad abacos, et sibi mutuo (accito si placet amico) dicere: c a l c u l e m u s". Podobne stwierdzenia znajdują się w innych tekstach cytowanego tomu, np. s. 64-65, 125. CALCULEMUS jest dzisiaj nazwą międzynarodowej grupy badawczej. Celem tej grupy jest rozwój nowej generacji systemów matematycznego wspomagania opartych na integracji mocy dedukcyjnej systemów dedukcyjnych i mocy obliczeniowej algebraicznego systemu komputera. Zob. http://www.calculemus.net/

> Wenn Gott Eurer Hochfürstl. Durchlaucht noch den Gedanken eingäbe, mir lediglich zu bewilligen, daß die 1200 Taler, die festzusetzen Ihr die Güte hattet, zu einer Dauerrente würden, so wäre ich ebenso glücklich wie Raymund Lull[12], und vielleicht mit größerem Recht. […] Denn meine Erfindung umfasst den Gebrauch der gesamten Vernunft, einen Richter für alle Streitfälle, einen Erklärer der Begriffe, eine Waage für die Wahrscheinlichkeiten, einen Kompass, der uns über den Ozean der Erfahrungen leitet, ein Inventar der Dinge, eine Tabelle der Gedanken, ein Mikroskop zum Erforschen der vorliegenden Dinge, ein Teleskop zum Erraten der fernen, einen generellen Calculus, eine unschädliche Magie, eine nicht-chimärische Kabbala, eine Schrift, die jedermann in seiner
>
> Sprache liest; und sogar eine Sprache, die man in nur wenigen Wochen erlernen kann und die bald in der ganzen Welt Geltung haben wird. Und die überall, wo sie hinkommt, die wahre Religion mit sich bringt.

Leibniza idea maszynowego ,,myślenia" zapisanego językiem binarnym w jakiejś części realizowana jest przez współczesną informatykę. Zastosowania informatyki zmieniają nasze życie tak, jak tego chciał Leibniz, gdy pisał, że będzie to (*characteristica universalis*) ostatnim wysiłkiem ludzkiego ducha, gdy bowiem projekt zostanie zrealizowany, będzie miał człowiek narzędzie powiększające możliwości rozumu tak jak teleskop, który uzdalnia widzenie, i mikroskop, który umożliwił ujrzenie wnętrza przyrody. Dzięki niemu [29: 373-381 – Leibniz an Heinrich Oldenburg 1673-1676]:

> … w trakcie mówienia, samą mocą sformułowań, gdy język będzie prowadził umysł, nawet głupcy będą wygłaszać wielce inteligentne zdania, dziwując się sami swojej wiedzy, bez trudu pokonując swą umysłową niemoc, a będzie owe wypowiedzi rozumiał nawet ktoś najgłupszy[13].

Przychodzi nam dziś dokonać osądu, do którego wzywał Leibniz, gdy pisał [tamże]:

> Osądź, jak wielkie będzie nasze szczęście, jeśli za sto lat od tej chwili język taki powstanie[14].

---

[12] Pomysły Lullusa (*Ars Magna et Ultima*) zainspirowały wielu. Werner Künzel [25] pisze: „Od 1987 programowałem ten pierwszy piękny algorytm historii filozofii w językach komputerowych *COBOL*, *Assembler* oraz *C*".

[13] „… inter loquendum ipsa phrasium vi lingua mentem praecurrente praeclaras sententias effutient imprudentes, et suam ipsi scientiam mirantes, cum ineptiae sese ipsae prodent nudo vultu, et ab ignarissimo quoque deprehendentur". Przekład polski – W. Marciszewski.

[14] „Quantam nunc fore putas felicitatem nostram si centum ab hinc annis talis lingua coepisset". Przekład polski – W. Marciszewski.

Ziarna zostały zasiane[15]. Wysiłkiem i geniuszem Boole'a oraz Fregego, Peano, Peirce'a i wielu innych tworzone są podstawy rachunków logicznych. Wciąż nie mamy jednak dobrej definicji rachunku. Na przełomie wieków XIX i XX David Hilbert sformułował wielki program formalizacji matematyki. Hilbert uważa, że obowiązuje ogólna zasada odnosząca się do rozumu ludzkiego, że każde zagadnienie, które rozum stawia, może być przez rozum rozwiązane. Hilbert odrzuca słynne hasło E. Dubois-Raymonda: „I g n o r a m u s et i g n o r a b i m u s"[16].

> Wewnętrzny głos mówi: jest problem, szukaj rozwiązania. Znaleźć możesz je za pomocą czystego myślenia; albowiem w matematyce nie istnieje *Ignorabimus*!

Stanowisko to przenosi na całość poznania naukowego. W wystąpieniu królewieckim z 1930 r., kiedy to otrzymywał honorowe obywatelstwo miasta swojego urodzenia, stwierdza, że cała nasza kultura, oparta na intelektualnym poszukiwaniu i wykorzystaniu przyrody, znajduje swoje podstawy w matematyce. Za Galileuszem głosi, że „tylko ci, którzy nauczyli się języka i znaków, którymi do nas przemawia przyroda, mogą przyrodę zrozumieć". Swoje wystąpienie kończy tak, jak to czynił już na II kongresie matematyków w Paryżu w 1900 r.: „W i r m ü s s e n w i s s e n. W i r w e r d e n w i s s e n" („Musimy wiedzieć. Będziemy wiedzieć"). Przekonanie to było znaczące dla postawy Hilberta. Tej treści inskrypcja znajduje się nawet na jego grobie. Żeby wiedzieć, musimy mieć narzędzie, które pozwoli nam tę wiedzę osiągnąć.

Podstawowym pojęciem programu Hilberta jest pojęcie rozstrzygalności. Niemieckie określenie tego problemu: *Entscheidungsproblem* na stałe weszło do języka nauki[17]. Musimy mieć wiarygodną i obiektywną metodę, która pozwala w skończonej liczbie mechanicznych kroków dać odpowiedź na pytanie, czy dana formuła jest twierdzeniem.

Uprawnionymi metodami rozwiązywania problemów matematycznych są dedukcja i ścisłość. Rozwiązanie problemu ma być oparte o skończoną liczbę przesłanek i skończoną liczbę wnioskowań. Przesłanki winny być sformułowane precyzyjnie. Jest to postulat ścisłości dowodzenia. W dowodzeniu nieodzowne są przedstawienia geometryczne. Znaki arytmetyczne to zapisane figury geometryczne, a figury geometryczne to narysowane formuły

[15] Więcej na temat Leibniza jako prekursora informatyki zob. [43, 42].

[16] Emil du Bois-Reymond (1831-1889), niemiecki fizjolog, wypowiada je w *Über die Grenzen des Naturerkennens*.

[17] Zagadnieniu rozstrzygalności poświęciłem więcej uwagi w pracy [41].

i żaden matematyk nie może się bez nich obejść. Hilbert zatem dopuszcza rozstrzyganie tylko za pomocą rachunków. Pierwszą kwestią w *Entscheidungsproblem* było znalezienie precyzyjnego matematycznego sformułowania metody.

Po pracach Gödla, Churcha i Turinga program Hilberta legł w gruzach. Paradoksalnie prace te są fundamentalne dla informatyki. Church i Turing, niezależnie od siebie, podali definicję rachunku. Obie okazały się równoważne. Zdaniem Churcha przewagę ma definicja korzystająca z pojęcia maszyny (Turinga), podana przez Turinga.

Turing sformułował pojęcie maszyny pracującej na taśmie papierowej, na której byłyby drukowane symbole, i zapytał, jaki najbardziej ogólny typ procesów może być wykonany przez taką maszynę. Pokazał, że jest to równoważne temu, co zawsze może być osiągnięte przez kogoś posługującego się zbiorem instrukcji logicznych. Ma tu miejsce potrójna odpowiedniość: między logicznymi instrukcjami, działaniem umysłu i maszyną, która w zasadzie mogłaby być zrealizowana w fizycznej postaci[18].

Teza Churcha-Turinga jest matematycznym stwierdzeniem uniwersalności: każdy komputer z pewną minimalną pojemnością jest w istocie w stanie wykonać wszystkie zdania, jakie może wykonać jakikolwiek inny komputer. Wszystkie komputery, począwszy od telefonów komórkowych, a skończywszy na superkomputerach, mają możliwość wykonać te same zadania obliczeniowe, jeśli tylko będą miały wystarczająco dość czasu i odpowiednią pamięć. Inaczej mówiąc, teoretycznie maszyna Turinga może wykonać wszystko to, co może wykonać jakikolwiek komputer.

Zdaniem Gödla Turing pokazał równoważność pojęcia mechanicznej procedury ze swoją maszyną. Słowa Turinga: „Możemy porównać człowieka, który rachuje na liczbach rzeczywistych, do maszyny" współbrzmią z komentarzem Wittgensteina: „T e m a s z y n y t o l u d z i e, k t ó r z y r a c h u j ą".

Turing głosił:

[18] W 1999 r. „Time Magazine" uznał Alana Turinga za jednego ze stu najbardziej znaczących ludzi XX wieku ze względu na jego rolę w stworzeniu nowoczesnego komputera: „The fact remains that everyone who taps at a keyboard, opening a spreadsheet or a word-processing program, is working on an incarnation of a Turing machine" („Jest bezspornym faktem, że każdy, kto dotyka klawiatury, otwiera arkusz kalkulacyjny lub edytor tekstu, pracuje z wcieleniem maszyny Turinga").

W ankiecie przeprowadzonej przez BBC w 2002 r. Turing znalazł się na 21 pozycji w rankingu stu największych Brytyjczyków.

> Możemy mieć nadzieję, że ostatecznie maszyny będą mogły konkurować z ludźmi we wszystkich czysto intelektualnych dziedzinach.

Gödel i Turing wykazali, że marzenia Hilberta są nierealne. Postawienie problemu rozstrzygalności okazało się jednak najbardziej owocne ze wszystkich zadań, jakie Hilbert stawiał matematykom XX wieku. Pytanie to doprowadziło do maszyny Turinga i teorii języków formalnych, które stanowią podstawę współczesnej informatyki i technik komputerowych. Podsumowuje to Chaitin [6]:

> […] systemy formalne nie okazały się znaczące dla rozumowania, lecz odniosły wspaniały sukces w wypadku obliczania. Tak więc najbardziej niewiarygodny światowy sukces odniósł Hilbert w technologii a nie w epistemologii. […]
>
> Na koniec zacytuję Izajasza Berlina z właśnie opublikowanego pośmiertnego zbioru esejów *The Power of Ideas*: „Ponad sto lat temu niemiecki poeta Heine ostrzegł Francuzów, aby nie niedoceniali siły idei: filozoficzne pojęcia pielęgnowane w ciszy profesorskiego gabinetu mogą zburzyć cywilizację". Takie postrzeganie idei, jak myślę, jest rzeczywiście prawdziwe. Idea Hilberta osiągnięcia kresu, pełnej formalizacji, kierowana racjami epistemologicznymi, którymi były filozoficzne kontrowersje w sprawie podstaw matematyki — czy są podstawy? Oraz sposób, w jaki projekt upadł, jak wyjaśniłem, z powodu prac Gödla i Turinga. Tu jednak mamy pełną formalizację, jaką są komputerowe języki programowania, one są wszędzie!

Podstawy teoretyczne współczesnej informatyki zrodziły się w pracach Kurta Gödla, Alonzo Churcha i Alana Turinga. Narodziny komputera w połowie lat czterdziestych zawdzięczamy praktycznemu geniuszowi Turinga oraz wizji i intelektualnej sile innego wielkiego logika matematycznego – Johna von Neumanna. Obaj oni, Turing i von Neumann, odegrali głowną rolę nie tylko w zaprojektowaniu i zbudowaniu pierwszych komputerów, lecz również położeniu logicznych fundamentów pod rozumienie procesów obliczeniowych i rozwój formalizmów informatycznych.

Podsumowując dotychczasowe rozważania o podstawach teoretycznych informatyki można powiedzieć: n a p o c z ą t k u b y ł a l o g i k a.

2. LOGIKA JĘZYKIEM INFORMATYKI

Logika w informatyce odgrywa rolę podobną do roli analizy matematycznej i równań różniczkowych w fizyce i naukach inżynierskich. Galileuszowi przypisuje się tezę, że „księga natury napisana jest językiem matematyki".

Fizyka, a z nią inne nauki przyrodnicze, odniosła sukces, pisząc swoją księgę językiem analizy matematycznej. W tym kontekście naturalne jest pytanie o język informatyki. Odpowiada się, że to logika jest „the calculus of computer science".

W opinii Quine'a [37] jedno i drugie, logika i informatyka, mają wspólne podstawy:

> W pełni czysta teoria dowodu matematycznego i w pełni technologiczna teoria liczenia maszynowego są na jednym poziomie a podstawowe intuicje każdego są intuicjami drugiego.

Znaczenie logiki dla informatyki wzrasta wraz z jej rozwojem zarówno, gdy chodzi o pojęcia, jak i metody. Logika w większym stopniu wplata się w informatykę niż w matematykę. Stąd też wiele dociekań logicznych inspirowanych jest problemami informatyki. Wzrost zainteresowania logiką w związku z rozwojem informatyki zapowiadał już Turing [45]:

> Przewiduję, że koniec końców cyfrowe maszyny liczące będą stymulować znaczące zainteresowanie logiką symboliczną […] Język, w którym komunikujemy się z tymi maszynami […] jest pewnym typem logiki symbolicznej.

Logika pierwszego rzędu jest fundamentem nowoczesnych systemów bazodanowych, a standardowe języki wyszukiwania, takie jak SQL (**S**tructured **Q**uery **L**anguage) oraz QBE (**Q**uery-**B**y-**E**xample), są syntaktycznymi wariantami języka tej logiki. Mocniejsze języki zapytań oparte są na rozszerzeniach języka logiki pierwszego rzędu z rekurencją i są reminiscencją dobrze znanych wyszukiwań stało punktowych, badanych w teorii skończonych modeli. Podobnie ma się sprawa w wypadku danych webowych. Wpływ logiki na bazy danych jest jednym z najbardziej uderzających przykładów efektywności logiki w informatyce.

Problematyka sztucznej inteligencji jest „wspólnym" działem logiki i informatyki teoretycznej. John McCarthy, laureat Nagrody Turinga za wkład w rozwój Sztucznej Inteligencji, twórca samego terminu „Artificial Intelligence" (Sztuczna Inteligencja), w 1960 r. powiedział, że:

> rozsądna jest nadzieja, że w najbliższym stuleciu relacja między obliczaniem a logiką matematyczną będzie równie owocna jak w ostatnim stuleciu między analizą matematyczną a fizyką.

Gödel był przekonany, że „ludzki duch nie jest zdolny do zmechanizowania wszystkich swoich intuicji matematycznych". W 1989 r. fizyk Roger Penrose w książce *The Emperor's New Mind: Concerning Computers, Minds,*

and The Laws of Physics[19] (początek tytułu rozumieją ci, którzy pamiętają baśnie Andersena), odwołując się do mechaniki kwantowej, dowodzi, że ludzki umysł jest w stanie wykroczyć poza dedukcyjne rozumowanie, a zatem żadna maszyna nie będzie w stanie symulować jego myślenia.

W sensie podstawowym logika jest nauką o językach sformalizowanych i rozumowaniu, informatyka zaś podejmuje podobne problemy, mając dodatkowe zadanie wyrażenia tych formalizacji w sposób techniczny przez stworzenie mechanizmów, które działają zgodnie z założonymi regułami. To w istocie doprowadziło do niedawnego zastosowania informatyki w badaniach nad logiką w sposób eksperymentalny.

3. LOGIKA NARZĘDZIEM INFORMATYKI

Komputer działa, wykonując zadany program. Poprawność działania zależy od poprawnej konstrukcji samego urządzenia i od poprawności programu, czyli poprawności systemu informatycznego. Jak zauważa Henk Barendregt: „It is fair to state, that in this digital era correct systems for information processing are more valuable than gold”[20]. Problem fizycznych wad sprzętu i ich wykrywania – jako, że jest to problem inżynierski – zostanie tu pominięty. Kwestia poprawności systemów informatycznych jest problemem, który może być rozwiązywany za pomocą testowania lub za pomocą narzędzi formalnych. Testowanie może nie być możliwe, np. ze względów etycznych, może być ekonomicznie bardzo kosztowne, a co najważniejsze – z zasady nie jest pełne, a więc nie jest wykluczona wada, która ujawni się w innej niż w testowanych sytuacjach. Test może także nie dawać wskazówki, co jest źródłem błędu. Narzędzia logiczne unikają wszystkich tych ograniczeń. W ich wypadku podstawowym problemem są ograniczenia na czas działania programu i pamięć komputera użytego do weryfikacji. Podstawową korzyścią z użycia metod formalnych jest redukcja błędów systemu. W konsekwencji ich podstawowymi obszarami zastosowań są systemy krytyczne: informatyczne systemy kontroli lotów, systemy sygnalizacji kolejowej, systemy rakietowe, medyczne systemy kontroli. Ze względu na bezpieczeństwo metody formalne są zalecane dla systemów krytycznych na przykład

[19] Wydanie polskie: R. P e n r o s e, *Nowy umysł cesarza: o komputerach, umyśle i prawach fizyki*, przeł. P. Amsterdamski, Warszawa 1995.

[20] Zob. H.P. B a r e n d r e g t, *The Quest for Correctness*, http://repository.ubn.ru.nl/bitstream/2066/17273/1/13361.pdf

przez *IEC* (**I**nternational **E**lectrotechnical **C**ommission), *ESA* (**E**uropean **S**pace **A**gency). Raport przygotowany przez *FAA* (**F**ederal **A**viation **A**uthority) oraz NASA (**N**orth-**A**tlantic **S**pace **A**gency) stwierdza:

> Formalne metody powinny być elementem kształcenia każdego informatyka i programisty, tak jak odpowiednia dziedzina matematyki stosowanej jest niezbędnym elementem edukacji inżynierów.

Szczególne miejsce w metodach formalnych zajmuje logika temporalna. Tę nową rolę logiki komentuje M. Davis [12]:

> Kiedy byłem studentem, nawet topologowie uważali logików matematycznych za ludzi oderwanych od rzeczywistości. Dzisiaj związki między logiką a komputerami są obszarem praktyki inżynierskiej na każdym poziomie organizacji komputera.

Logicy to „abstrakcyjni inżynierowie" (w dobrym znaczeniu tego wyrażenia). W ocenie Moshe Vardi ma miejsce ciągle wzrastająca interakcja[21] jednego i drugiego, informatyki i logiki. Dodajmy, że dobrym przykładem tej interakcji jest logika temporalna: jej powstanie inspirowały problemy filozoficzne, a swój dzisiejszy rozwój zawdzięcza informatyce.

Okazuje się, że informatyka w większym stopniu czyni praktyczny użytek z logiki formalnej niż matematyka, która jej rozwój inspirowała [23]:

> Do czasu wynalezienia cyfrowego komputera było niewiele zastosowań formalnej matematycznej logiki poza badaniem samej logiki. W szczególności, kiedy wielu logików badało alternatywne systemy dowodzenia, studiowało moce różnych logik oraz formalizowało podstawy matematyki, niewielu ludzi używało logiki formalnej i formalnych dowodów do analizy własności innych systemów. Brak zastosowań może mieć dwa powody: (i) sam formalizm formalnej logiki pomniejsza jej jasność jako narzędzia komunikowania i rozumienia, oraz (ii) „naturalne" zastosowania logiki matematycznej w okresie pre-cyfrowego świata były w czystej matematyce i było niewielkie zainteresowanie formalizacją jako wartością dodaną. Oba te powody zmienił wynalazek cyfrowego komputera. Uciążliwe i precyzyjne manipulowanie formułami w formalnej syntaktyce może być przeprowadzone przez program działający pod kierunkiem użytkownika, który generalnie zajęty jest bardziej strategią dowodzenia.

Dodajmy tu, że możliwości, jakie stwarza komputer w przetwarzaniu formalnego zapisu, wykorzystują różne projekty automatycznego dowodzenia (*prover*) lub automatycznego sprawdzania poprawności dowodu (*proof checker*).

[21] Zob. www.ii.uni.wroc.pl/cms/pl/node/934

Logicy zainspirowani zagadnieniami informatycznymi i sami informatycy wnieśli do logiki nową problematykę oraz stworzyli nowe i udoskonalili znane metody. W logice Hoare'a (Floyda-Hoare'a) program rozumiany jest jako przejście od stanu początkowego do stanu końcowego. Takie podejście nie jest odpowiednie w wypadku programów bez stanu końcowego. Logika ta nie radzi sobie więc z systemami reaktywnymi i bez stanów końcowych, takimi jak systemy operacyjne, protokoły, programy współbieżne, systemy sprzętowe. Klasyczne metody odzwierciedlają statyczną naturę pojęć matematycznych. Dynamiczne działanie programu wymaga innego podejścia. Logika temporalna i jej język są szczególnie interesujące w wypadku systemów reaktywnych i współbieżnych. Z jednej strony język specyfikacji powinien być prosty i łatwy do zrozumienia tak, by mógł być używany przez kogoś, kto nie jest ekspertem. Z drugiej strony powinien być wystarczająco ekspresywny, aby formalizować procesy przechodzenia od jednego do drugiego stanu oraz ich interakcję. Ponadto powinien posiadać formalną semantykę, która oddaje w sposób jednoznaczny intuicyjne znaczenie języka rozkazów.

Logikę modalną z problemami informatyki kojarzy w 1974 r. Burstall. Pratt, twórca Dynamicznej Logiki Programów, o zainteresowaniu się logiką modalną pod kątem jej zastosowań w informatyce pisze [33]:

> Wiosną 1974 r. prowadziłem zajęcia z semantyki i aksjomatyki języków programowania. Zasugerowany przez jednego ze studentów, R. Moore'a, rozważyłem zastosowanie logiki modalnej do formalnego badania konstrukcji pochodzącej od C.A.R. Hoare'a, „$p\{a\}q$", która wyraża myśl, że jeśli p zachodzi przed wykonaniem programu a, to po tym ma miejsce q. Chociaż początkowo byłem sceptyczny, to weekend z Hughes'em i Cresswellem przekonał mnie, że możliwy jest bardzo harmonijny związek między logiką modalną a programami. Związek zapowiadał się interesująco dla informatyków ze względu na moc oraz matematyczną elegancję ujęcia. Wydawał się także mieć znaczenie dla logików modalnych z powodu dobrego umotywowania i potencjalnie owocnego związku między logiką modalną a Tarskiego rachunkiem relacji binarnych.

Podejście wykorzystujące logikę modalną było istotnie lepsze od podejścia opartego na mechanizmie pre-warunek/post-warunek, jaki zakłada logika Hoare'a[22]. Modele Kripkego, standardowa struktura semantyczna, w której

[22] Pod koniec lat sześćdziesiątych XX wieku Floyd, Hoare i Naur proponowali aksjomatyczne dowodzenie własności programów sekwencyjnych ze względu na ich specyfikację. E.G. Dijkstra rozszerzył ideę Hoare'a. Metoda teorio-dowodowa była już zastosowana przez Turinga w jego pracach nad poprawnością programów.

interpretowane są języki modalne nie są niczym innym niż grafami. Grafy są powszechne w informatyce.

Związek między możliwymi światami logików a wewnętrznymi stanami komputera jest łatwy do opisania. W semantyce możliwych światów, φ (możliwe, że φ) jest prawdziwe w pewnym świecie w wtedy i tylko wtedy, gdy φ jest prawdziwe w pewnym świecie w' osiągalnym z w. Twierdzenia logiki modalnej zależą od własności relacji osiągalności, jak np. zwrotność, symetryczność itd. Relacja osiągalności logiki modalnej może więc być rozumiana jako relacja między stanami komputera, kontrolowanego przez taki program, że maszyna, rozpoczynając działanie w jakimś stanie, (po skończonym czasie) znajdzie się w jednym z osiągalnych stanów. Na przykład w wypadku pewnych programów nie można powrócić z jakiegoś stanu do stanu wcześniejszego; stąd w takim wypadku osiągalność nie jest symetryczna.

Struktury Kripkego są modelami logik modalnych inspirowanych zagadnieniami logicznymi i filozoficznymi. W informatyce struktury Kripkego kojarzone są z systemami przejść. W logikach modalnych funkcję podobną do roli struktur Kripkego pełnią jeszcze struktury Hintikki. Rama Kripkego składa się z niepustego zbioru oraz binarnej relacji zdefiniowanej w tym zbiorze. W logice modalnej elementy tego zbioru są nazywane możliwymi światami, a relacja między nimi jest rozumiana jako relacja osiągalności jednego świata z drugiego. W szczególności, kiedy relacja osiągalności interpretowana jest temporalnie otrzymujemy modalności temporalne, które są przedmiotem logiki temporalnej. Logika temporalna stosowana w informatyce bazuje na czasie obliczeniowym, czyli czasie wyznaczonym przez następstwo stanów komputera cyfrowego. Model Kripkego może być opisany jako etykietowany graf przejść stanowych, gdzie wierzchołki są etykietowanymi zbiorami zdań atomowych i nazywane są stanami. Krawędzie oznaczają przejścia.

Problem wykorzystania logiki temporalnej do rozumowań programistycznych podejmuje Kröger. Rozwój badań nad logiką temporalną pod kątem jej zastosowań informatycznych wiąże się z pracami Amira Pnueli[23]. Zainspirowała go lektura *Temporal Logic* Reschera i Urquharta. Jego praca *The temporal logic of programs* [31] traktowana jest jako klasyczne źródło logiki

[23] W 1996 r. A. Pnueli otrzymał nagrodę Turinga (*The Turing Award*) za inspirujące prace wprowadzające logikę temporalną do informatyki i za wybitny wkład do weryfikacji programów i systemów.

temporalnej dla specyfikacji i weryfikacji programów, w szczególności dla programów współbieżnych[24]. Powszechnie przyjmuje się, że była przełomowa, jeśli chodzi o zastosowanie logiki temporalnej w informatyce. A. Pnueli i Z. Manna dostrzegli rosnące znaczenie systemów reaktywnych[25], a logikę temporalną uznali za odpowiednią dla ich formalnej[26] specyfikacji i weryfikacji[27]. W szczególności stwierdzono, że modalności temporalne nadają się do formułowania tak znaczących własności, jak sprawiedliwe szeregowanie, brak zakleszczeń i wielu innych.

Zdaniem Amira Pnueli logika temporalna może być zastosowana jako formalizm rozumowania o działaniu programów komputerowych, w szczególności systemów współbieżnych[28]. Według Bochmanna [7: 1]:

> logika temporalna dostarczała bardziej eleganckiego sposobu mówienia o żywotności i ewentualności; w społeczności zajmującej się weryfikacją protokołów mówi się o osiągalnych stanach zakleszczenia (łatwe do charakteryzacji) lub o niepożądanych pętlach (trudne do charakteryzacji).

Najogólniej biorąc, ważnymi własnościami operacji są poprawność i bezpieczeństwo. Dla Edmunda E. Clarke'a [7: 1] prace Pnueli, Owickiego oraz Lamporta:

---

[24] Dla zapewnienia poprawności działania takich programów niezbędna jest specyfikacja współdziałania różnych procesorów. Terminarz działań musi być tak dopracowany, aby zachowana była integralność informacji, które rozdzielone są na różne procesory. Podstawowe jest rozróżnienie między „żywotnościowymi" własnościami o postaci $F\varphi$, co zapewnia otrzymanie pożądanych stanów w procesie obliczeń, a własnościami „bezpiecznymi" postaci $G\varphi$, co zapewnia, że nie otrzymamy stanów niepożądanych.

[25] Systemy reaktywne, takie jak systemy operacyjne oraz systemy czasu rzeczywistego – w odróżnieniu od programów, których wynik działania jest w pełni określony przez dane wejściowe oraz warunki początkowe i końcowe – w czasie wykonywania wchodzą w interakcję z otoczeniem i przebieg tej interakcji oraz zdarzenia systemowe są istotne. Systemy te mają wielorakie i wciąż coraz większe zastosowanie. Obejmują m.in. systemy zagnieżdżone, systemy kontroli procesów, systemy współbieżne oraz rozproszone.

[26] Termin „formalny" bywa mylony ze „ścisły". Zauważmy jednak, że to, co formalne, jest ścisłe, lecz nie na odwrót.

[27] Istnieje wiele technik formalnej specyfikacji. Języki tych specyfikacji różnią się znacznie, zarówno ze względu na syntaktykę jak i semantykę oraz ze względu na obszary zastosowania. Wyróżnia się specyfikację logiczną, techniki algebry procesów, techniki oparte na automatach, sieciach Petriego, języki synchroniczne. Do grupy języków bardziej zorientowanych inżyniersko należy np. język Z.

[28] System współbieżny to taki system, którego działanie jest wynikiem interakcji i ewolucji wielokrotnych agentów obliczających. Interaktywne procesy nie wiedzą o wewnętrznych stanach innych. Początkowe zainteresowanie systemami współbieżnymi było motywowane zwiększeniem prędkości, co dawały komputery wieloprocesorowe.

przekonywująco pokazują, że Logika Temporalna jest idealna dla wyrażania pojęć takich, jak wzajemne wykluczanie, brak zakleszczenia oraz brak zagłodzenia.

Dzisiaj logika temporalna zdaje się być najbardziej badana i rozwijana w związku z jej zastosowaniami w praktyce informatycznej.

Logicy różnią się od informatyków w podejściu do logiki [5: 315]:

Rozstrzygalność i aksjomatyzowalność są standardowymi problemami logików; dla praktyków, tym ważnym pytaniem jest sprawdzanie modelowe.

Zdaniem Dijkstra[29]:

Sytuacja programisty jest podobna do sytuacji matematyka, który rozwija teorię i dowodzi twierdzeń. […] Nigdy nie może on gwarantować, że dowód jest poprawny, co najwyżej może powiedzieć: „Nie znalazłem żadnych błędów". […] Jest to tak niezmiernie wiarygodne, że analogia może służyć jako wielkie źródło inspiracji. […]

Nawet przy założeniu bezbłędnie działającej maszyny powinniśmy sobie zadawać pytania: „Gdy automatyczny komputer wytwarza wyniki, dlaczego ufamy im, jeśli tak jest?", a następnie: „Jakimi środkami możemy wzmocnić nasze przeświadczenie, że wytworzone wyniki są rzeczywiście wynikami zamierzonymi?"

W innej pracy [14: 6] Dijkstra pisze:

Program testujący może być użyty do pokazania obecności pluskwy[30], lecz nie do pokazania jej nieobecności.

Chociaż więc teoria mówi o weryfikacji, jednak zauważamy, że prawdziwa jej wartość leży w falsyfikacji.

Według Emersona ze względu na niepełność testowania szukano alternatywnych metod. Najbardziej obiecujące okazało się podejście oparte na fakcie, że programy, a bardziej ogólnie – systemy informatyczne, mogą być postrzegane jako przedmioty matematyczne w zasadzie z dobrze określonym działaniem. To daje możliwość specyfikacji za pomocą logiki matema-

[29] Zob. E.W. D i j k s t r a, *Programming Considered as a Human Activity*, http://www.cs.utexas.edu/users/EWD/transcriptions/EWD01xx/EWD117.html

[30] W języku angielskim na określenie błędu w działaniu systemu informatycznego stosowane jest słowo „bug", co znaczy 'pluskwa, owad, robactwo'. Użycie tego słowa na niewytłumaczalną wadę urządzenia, również mechanicznego, od dawna należało do inżynierskiego żargonu. Z jego użyciem spotkamy się w liście napisanym w 1878 roku przez Th. Edisona. Słowo to stosowała również obsługa radarów w czasie II wojny światowej. Tworzona spontanicznie historia informatyki przypisuje pomysł tego terminu Grace Hopper. 9 września 1947 r. komputer Mark II działał niepoprawnie. Operator stwierdził, że powodem jest obecność ćmy. Ćma po wyjęciu została załączona do dziennika obsługi z dopiskiem: „First actual case of bug being found".

tycznej tego, czym jest to zamierzone (poprawne) działanie. Można zatem próbować dać formalny dowód lub inaczej ustalić, czy ten program odpowiada specyfikacji. Takie sposoby postępowania określane są zwykle jako metody formalne.

Problem weryfikacji, można powiedzieć, zrodził się w tym samym czasie co sama informatyka. Sformułowany w terminach Maszyny Turinga był rozważany już w pracy Turinga o problemie stopu [44]. Udowodnił on, że nie ma ogólnej metody dowodzenia poprawności programu. Dlatego też ta publikacja w jakiś sposób zakończyła ten problem. Zostało bowiem pokazane, że to zadanie nie jest teoretycznie mechanizowalne. Tak więc bez jakiejś wiedzy o detalach danego produktu generalnie jedna rzecz śmiało może być założona: ma pluskwy[31]. Jak matematycy, mimo twierdzenia Gödla, nie przestają dowodzić twierdzeń, tak informatycy nie przestają traktować weryfikacji jako podstawowego zagadnienia swojej dziedziny. Jest to sen na jawie milionów informatyków: kompilator, który automatycznie wykrywa wszystkie pluskwy w kodzie [22].

Powstanie i rozwój logiki temporalnej wiążą się z zagadnieniami filozoficznymi. Była ona raczej przedmiotem zainteresowania filozofów i semiotyków niż informatyków. Już jednak Arthur Prior, twórca logiki temporalnej, był świadomy praktycznych korzyści, jakie mają jej badania dla reprezentacji następstwa czasowego w obwodach komputerowych [35: 46]. Aczkolwiek brak jakichś metafizycznych przesłanek do pojmowania czasu jako struktury dyskretnej, to usprawiedliwiał jej badania tym, że [34: 67]:

> mają zastosowanie w ograniczonych dziedzinach dyskursu, w których zainteresowani jesteśmy jedynie tym, co zdarzy się następnie w ciągu dyskretnych stanów, np. w działającym komputerze cyfrowym.

Rescher i Urquhart [38] także wskazywali zastosowania logiki temporalnej jako narzędzia w badaniach „zaprogramowanej sekwencji stanów, deterministycznej lub stochastycznej […]".

Logika może być tworzona aksjomatycznie lub semantycznie. W wypadku tego pierwszego podejścia podane są aksjomaty i reguły dowodu. W wypadku semantycznie określonej logiki mamy model i pojęcie spełniania w modelu. Owa dualność prowadzi do dwóch podejść: teorio-dowodowego i teorio-modelowego. Metody formalne mogą być podzielone na te dwie podstawowe kategorie:

[31] Dzisiaj szacuje się, że w programach dla przemysłu wskaźnik błędu wynosi od 0,5 do 5 błędów na tysiąc linijek kodu źródłowego bez komentarzy.

– dowodzenie twierdzeń i sprawdzanie poprawności dowodu,
– sprawdzanie modelowe.

Poprawność formalnego wyprowadzenia może być sprawdzana mechanicznie, lecz znalezienie dowodu wymaga doświadczenia i intuicji, wymaga udziału bardzo sprawnego logika matematycznego[32]. W okresie, kiedy te metody wprowadzono – we wczesnych latach osiemdziesiątych XX wieku – „ręczne" dowodzenie było dominującym sposobem weryfikacji. Wszystkie dowody komercyjnie interesujących twierdzeń przeprowadzone za pomocą systemów dowodzenia mają jedno wspólne: niezbędny jest znaczny udział eksperta i duży trud – mówi się o tygodniach pracy. Już C.A.R. Hoare argumentował, że formalne dowody poprawności są tylko jedną z wielu możliwych opcji na drodze do niezawodnych systemów informatycznych.

Sprawdzanie modelowe jest preferowaną techniką weryfikacji, jest procedurą bardziej efektywną i łatwą do zrozumienia. Istotnym jednak powodem jest możliwość jej automatyzacji. Chociaż współczesne programy do sprawdzania modelowego (*model checker*) dają się zastosować do bardzo dużych przestrzeni stanów, np. 10^{120}, to jednak nie znaczy, że te stany są zbadane explicite. Na przykład twierdzenie o *FDIV*[33] wymaga sprawdzenia około 10^{30} przypadków. Ponieważ nie ma tu sposobu redukcji, trzeba by było, aby jeden przypadek był sprawdzany w czasie jednej femtosekundy (10^{-15} sekundy), a mimo to sprawdzenie twierdzenia zajęłoby więcej niż 10^7 lat [23: 183].

Zainteresowanie sprawdzaniem modelowym za pomocą logiki temporalnej zaczęło się na początku lat osiemdziesiątych XX wieku. Zarówno idea automatycznej weryfikacji programów współbieżnych, jak i termin *model checking* pojawiły się w pracy Clarke'a i Emersona [8]. Niezależnie podjęta była też przez Queille'a i Sifakisa [36][34]. Idea była rozwijana w pracach

[32] Różnicę między dowodzeniem a sprawdzaniem zauważał już Arystoteles, gdy mówił, że gdy ktoś poda dowód, to może sprawdzić jego poprawność. Jeśli zaś będzie miał dowód przeprowadzić, to nie zawsze to będzie potrafił zrobić.

[33] Chodzi o słynny błąd Intel Pentium *FDIV* z 1994 r., w wyniku którego *Intel* stracił 475 milionów dolarów [10].

[34] E.M. Clarke i E.A. Emerson zinterpretowali współbieżny system jako skończoną strukturę Kripkego/system przejść, a własności wyrażone były w języku logiki *CTL* (**C**omputational **T**ree **L**ogic), logiki drzew obliczeniowych. J.-P. Queille i J. Sifakis oparli się na sieciach Petri a własności były wyrażone w języku logiki czasu rozgałęzionego. Edmund M. Clarke, jr. (CMU, USA), Allen E. Emerson (Texas at Austin, USA), Joseph Sifakis (IMAG Grenoble, France) zostali laureatami nagrody Turinga za 2007 r. W uzasadnieniu czytamy: „For their roles in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries".

Clarke'a, Emersona, Sistla i innych. Istotny impet sprawdzaniu modelowemu nadał Kenneth L. McMillan, który – stosując wprowadzoną i spopularyzowaną przez Randy Bryanta technikę diagramów binarnych decyzji, BDD (**B**inary **D**ecision **D**iagrams) – rozwinął wysoce efektywne Symboliczne Sprawdzanie Modelowe.

Ograniczenie do systemów skończenie stanowych znaczy, że procedury sprawdzania modelowego są w istocie algorytmiczne, a w praktyce efektywne dla wielu systemów o znaczącym wymiarze.

Według Emersona [16: 31]:

> […] logika temporalna jest kluczowym czynnikiem w sukcesie sprawdzania modelowego. Mamy jedną strukturę logiczną z kilkoma operatorami temporalnymi dopuszczającymi wyrażenie nieograniczonych specyfikacji. Związek z językiem naturalnym jest również często znaczący. Logika temporalna umożliwia, ogólnie mówiąc, wyrazić własności poprawnościowe, które należy wyrazić. Bez tej zdolności, nie byłoby racji dla stosowania sprawdzania modelowego. W wielu wypadkach mogą być zastosowane alternatywne formalizmy temporalne, gdy mogą być bardziej ekspresywne lub zwięzłe niż logika temporalna. Jednak historycznie to logika temporalna była siłą napędową.

Za Emersonem powtórzmy pewne interesujące uwagi dotyczące sprawdzania modelowego [16: 41]:

> Edsger W. Dijkstra wyjaśniał mi, że jest to ,,akceptowalna kula (inwalidzka)", gdy dokonuje się weryfikacji po-fakcie[35]. Kiedy miałem przyjemność spotkać Saula Kripkego i objaśnić mu sprawdzanie modelowe nad strukturami Kripkego, skomentował to mówiąc, że nigdy nie myślał o tym. Daniel Jackson zauważył, że sprawdzanie modelowe ,,uratowało reputację" metod formalnych.

W kontekście naszych rozważań trzeba wspomnieć o Charlesie Leonardzie Hamblinie (1922-1985), który wprowadził do informatyki notację Łukasiewiczowską[36]. Nie był to przypadek. Uczył się logiki od Priora, który był wielkim zwolennikiem notacji Łukasiewiczowskiej i stosował ją, mimo że nie przysparzało mu to czytelników. Prior jako swoje lektury logiczne wskazuje prace logików polskich, m.in. Bocheńskiego[37]. Hamblin, będąc uczniem

[35] Ten typ weryfikacji ma następujące etapy:
- System jest budowany za pomocą standardowych metod tworzenia programu.
- Formalna specyfikacja pisana jest dla w pełni zbudowanego systemu.
- Dowodzi się własności specyfikacji i/lub kodu.

Innymi metodami weryfikacji są weryfikacja równoległa oraz podejście zintegrowane.

[36] W literaturze anglojęzycznej znana jako *Polish notation*.

[37] Na temat związków A.N. Priora z polskimi logikami zob. [40].

Priora, twórcy logiki temporalnej, był też prekursorem zastosowania logiki temporalnej w informatyce[38].

Wytworzony system informatyczny powinien spełniać warunki specyfikacji. To, czy tak jest, podlega weryfikacji[39]. W standardowym programie weryfikacyjnym niezbędne są trzy pojęcia. Potrzebne są:

- język do reprezentacji algorytmu,
- język specyfikacji dla wyrażenia właściwości weryfikowanego algorytmu oraz
- relacja spełniania dla weryfikacji poprawności algorytmu.

W związku z tym niezbędna jest formalna semantyka, aby jasno określone zostały działanie i specyfikacja. Znaczenie ma ekspresywność języka, a więc możliwości wyrażania specyfikacji oraz złożoność procedur rozstrzygających.

Zwykłe języki programowania nie są wystarczająco ekspresywne, aby wyrazić tak złożone właściwości systemów, jak współbieżność, niezdeterminowanie, synchronizacja procesów oraz ograniczenia na zdarzenia (przejścia) w takich systemach. Opisane zadania może wypełnić język logiki temporalnej. Operatory temporalne okazały się użyteczne do opisu działania systemów. Struktura stanów (sekwencja lub drzewo) są kluczowymi poję-

[38] Jego najbardziej znanym wkładem do filozofii jest praca *Fallacies* [19], która nawet dzisiaj jest standardową pracą o sofizmatach. Inną ważną pracą logiczną jest wydana już po śmierci książka *Imperatives* [20], która ma znaczenie w dzisiejszych pracach informatycznych, w tworzeniu protokołów dla delegowania zadań między programowymi agentami. Więcej na temat wkładu Hamblina do informatyki zob. [1, 3].

[39] Niektórzy autorzy rozróżniają między Walidacją (*Validation*) i Weryfikacją (*Verification*). Sprawdzanie w całości jest wówczas skrótowo oznaczane jako V&V. Walidacja odpowiada na pytanie: Czy to, co próbujemy robić, jest tym, o co na chodzi? (Czy to, co tworzymy, jest właściwą rzeczą?). Weryfikacja (funkcjonalna poprawność) odpowiada na pytanie: Czy to, co zrobiliśmy jest tym, co chcieliśmy zrobić? (Czy tworzymy tę rzecz poprawnie?) Metody weryfikacji mają na celu ustalenie, czy implementacja spełnia specyfikację ([2], s. 13). Różne charakterystyki weryfikacji i walidacji pochodzą od Boehma ([4]).

W ogólnej metodologii nauk termin „weryfikacja" oznacza stwierdzenie poprawności. Termin „falsyfikacja" (lub „refutacja") jest używany w znaczeniu 'wykrycie błędu'. W informatyce „weryfikacja" obejmuje oba znaczenia i odnosi do obustronnego procesu określenia, czy system jest poprawny czy błędny.

Dla Dijkstry ([15]) problem weryfikacji jest różny od problemu sympatyczności (*pleasantness*), który dotyczy tego, czy specyfikacja ujmuje system, którego naprawdę potrzebujemy i chcemy.

Emerson ([16], s. 28) zauważa, że: „Problem sympatyczności jest z natury pre-formalny. Pomimo to uznaje się, że staranne napisanie formalnej specyfikacji (która może być koniunkcją wielu subspecyfikacji) jest znakomitym sposobem oświecenia mroków związanych z problemem sympatyczności.

ciami, dzięki którym logika temporalna nadaje się do specyfikacji systemów informatycznych.

Język logiki temporalnej spełnia trzy ważne warunki:

– ma zdolność wyrażania wszystkich rodzajów specyfikacji (ekspresywność) niezależnie od języka programowania użytego w implementacji;
– ma rozsądną złożoność, aby specyfikować reguły (złożoność);
– dzięki podobieństwu do języka naturalnego jest łatwy do nauczenia się (pragmatyka).

Język logiki temporalnej może być zastosowany do specyfikacji szerokiego spektrum systemów informatycznych. Metody tej logiki mogą być zastosowane do weryfikacji [30]. W wypadku systemów reaktywnych logika temporalna jest bardziej użyteczna niż logika Floyda-Hoare, która jest lepsza w wypadku programów sekwencyjnych, czyli typu ,,wejście-wyjście". Język logiki temporalnej tworzy ogólną lingwistyczną ramę dedukcyjną dla systemu stanów w taki sam sposób jak logika klasyczna czyni to dla systemów matematycznych.

Każda weryfikacja korzystająca z techniki modelowej jest tylko tak dobra, jak dobry jest model. Jest to jedno z zastrzeżeń do paradygmatu sprawdzania modelowego, że wynik weryfikacji jest w takim stopniu godny zaufania, jaka jest dokładność modelu, który został skonstruowany dla analizy.

W sprawdzaniu modelowym zatem pierwszym zadaniem jest przełożenie systemu na formalny model akceptowany przez program sprawdzający (*model checker*). Jest to model w postaci struktury Kripkego lub etykietowanego grafu przejść stanowych. Powinien on adekwatnie opisywać działanie weryfikowanego systemu.

Drugim zadaniem jest specyfikacja własności, które musi posiadać realny system. Mechanicznie wspomagana weryfikacja własności złożonego systemu wymaga starannego opisana tych własności w języku formalnym z określoną semantyką. Ta specyfikacja jest zwykle podana w formalizmie logicznym. Ogólnie logiki temporalne są używane do temporalnej charakterystyki systemów.

Jeśli zastosowana jest logika temporalna, sprawdzanie modelowe polega na sprawdzaniu prawdziwości zbioru specyfikacji określonych za pomocą logiki temporalnej. Ogólnie rzecz ujmując, zastosowana logika temporalna to albo *CTL** lub jedna z jej podlogik *CTL* […] [9] bądź *LTL* […] [32].

Rozwijane są różne programy sprawdzające, które stosowane są do weryfikacji dużych modeli, systemów czasu rzeczywistego, systemów probabilistycznych, itd. – zob. [39]. Mimo utrudnień, jakie spowodowane są przez

eksplozję stanów, od samego początku sprawdzanie modelowe ma istotny wpływ na wysiłki w zakresie weryfikacji programów.

Weryfikacja za pomocą sprawdzania modelowego uzyskała popularność w przemyśle z dwóch powodów:

– procedura może być całkowicie zautomatyzowana oraz
– kontrprzykłady są generowane automatycznie, jeśli weryfikowana własność nie ma miejsca.

Warto zwrócić uwagę na inne zastosowania sprawdzania modelowego. Są to rozumienie i analiza kontraktów prawnych, które są przede wszystkim zapisem działań [11]; analiza procesów w żywych organizmach [21]; procesy e-biznesowe, takie jak systemy rachunkowości i organizacji pracy [46]. Sprawdzanie modelowe jest także stosowane w zadaniach sztucznej inteligencji, takich jak planowanie [17]. Również odwrotnie: techniki ze sztucznej inteligencji, mające odniesienia do planowania opartego na spełnialności, są odpowiednie dla sprawdzania modelowego [24].

Należy podkreślić, że sprawdzanie modelowe jest sposobem zarówno weryfikacji, jak i refutacji poprawności własności. Istotną siłą sprawdzania modelowego jest to, że może łatwo prowadzić do kontrprzykładu dla większości błędów [16]. Rzeczywistą wartością sprawdzania modelowego jest to, że jest wyjątkowo dobrym debuggerem – *reductio ad bug*[40].

Weryfikacja za pomocą sprawdzania modelowego jest bardziej sprawna w wypadku sprzętu niż programów. Programy są mniej ustrukturowane niż sprzęt. Na dodatek zwykle programy współbieżne są asynchroniczne, tj. większość działań wykonywanych przez różne procesy jest przeprowadzana niezależnie, bez globalnie synchronizującego zegara. Z tej racji dla programów sprawa eksplozji stanów staje się bardziej poważna. W konsekwencji sprawdzanie modelowe jest rzadziej używane do weryfikacji programów niż sprzętu ([7], s. 18).

W opinii Emersona [16] logika temporalna jest kluczowym czynnikiem sukcesu sprawdzania modelowego. Mamy jedną strukturę z kilkoma podstawowymi operatorami temporalnymi umożliwiającymi wyrażenie bez ograniczeń specyfikacji. Związki z językiem naturalnym są często bardzo znaczące. Logika temporalna daje możliwość, na ogół, wyrażania własności poprawnościowych, które powinny być wyrażone. Bez tej możliwości użycie sprawdzania modelowego nie miałoby racji. W niektórych wypadkach alternatywne temporalne formalizmy mogą być bardziej ekspresywne lub zwięzłe niż logika temporalna. Historycznie jednak logika temporalna była siłą napędową.

[40] Więcej na temat zalet i wad sprawdzania modelowego zob. [2].

Sprawdzanie modelowe tworzy pomost między informatyką teoretyczną a inżynierią sprzętową i inżynierią oprogramowania. Sprawdzanie modelowe nie wyklucza użycia metod teorio-dowodowych, i na odwrót: metody teoriodowodowe nie wykluczają użycia sprawdzania modelowego. Takie hybrydowe podejście ma wiele zalet. W praktyce metody te są komplementarne, przynajmniej na poziomie heurystycznym.

ZAKOŃCZENIE

Leibniz zamierzał stworzyć *lingua characteristica universalis* – język, w którym dałaby się zapisać wszelka wiedza (ekspresywność), oraz *calculus ratiocinator* – metodę umożliwiającą rachunkowe określenie prawdziwości dowolnego zdania tego języka. Taka idea leży u podstaw współczesnej logiki. Taka idea w odniesieniu do poprawności sprzętu i programów leży u podstaw metod logicznych weryfikacji systemów informatycznych.

Dziś widzimy, że dalszy rozwój informatyki zależy istotnie od postępu badań logicznych, a logika – ta starożytna dyscyplina – znalazła nowe bogate pole dociekań, zyskała nowe perspektywy badań. Jej zastosowania mogą mieć tak znaczący wymiar praktyczny, o jakim nie śniło się filozofom.

REFERENCJE

[1] Allen J. F. 1985: *Charles Hamblin (1922–1985)*, „The Australian Computer Journal" s. 194-195.

[2] Baier C., Katoen J.-P. 2008: *Principles of Model Checking*, Foreword by Kim Guldstrand Larsen, The MIT Press.

[3] Barton R.S. 1970: *Ideas for computer systems organization: a personal survey*, [w:] J. S. Jou (ed.), *'Software Engineering'. Proceedings of the Third Symposium on Computer and Information Sciences held in Miami Beach, Florida, December 1969*, Vol. 1, New York, NY: Academic Press, s. 7-16.

[4] Boehm B. W. 1981: *Software Engineering Economics*, Prentice-Hall.

[5] Bradfield J.C., Stirling C. 2001: *Modal logics and μ-calculi: An introduction*, [w:] J.A. Bergstra, A. Ponse, S.A. Smolka (eds), *Handbook of Process Algebra*, Elsevier Science, chapter 4, s. 293-330.

[6] Chaitin G.J. 2004: *Wissen und glauben/knowledge and belief*, [w:] W. Löffler, P. Weingartner (eds), *Akten des 26. Internationalen Wittgenstein-Symposiums 2003*, Wien: ÖBV & HPT, s. 277-286.

[7] Clarke E.M. 2008: *The birth of model checking*, [w:] DBLP:conf/spin/5000, s. 1-26.

[8] Clarke E.M., Emerson E.A. 1982: *Design and synthesis of synchronization skeletons using branching-time temporal logic*, [w:] *Logic of Programs, Workshop*, (Lecture Notes in Computer Science, Vol. 131), London, UK: Springer-Verlag, s. 52-71.
[9] Clarke E.M., Emerson E.A., Sistla A.P. 1986: *Automatic verification of finite-state concurrent systems using temporal logic specifications*, „ACM Transactions on Programming Languages and Systems" (2), s. 244-263.
[10] Coe T., Mathisen T., Moler C., Pratt V. 1995: *Computational aspects of the pentium affair*, „ IEEE Computational Science & Engineering" vol. 2, 1, s. 18-31.
[11] Daskalopulu A. 2000: *Model checking contractual protocols*, [w:] J. Breuker, R. Leenes, R. Winkels (eds), *'Legal Knowledge and Information Systems'*, JURIX 2000: The 13th Annual Conference, IOS Press, Amsterdam, pp. 35–47.
[12] Davis M. 1988: *Influences of mathematical logic on computer science*, [w:] *A half-century survey on The Universal Turing Machine*, New York, NY: Oxford University Press, s. 315-326.
[13] Davis M. 2000: *The Universal Computer: The Road from Leibniz to Turing*, New York: W.W. Norton & Company.
[14] Dijkstra E.W. 1968: *Notes on structured programming*, [w:] E.W.D.O.-J. Dahl, C.A.R. Hoare (eds), *Structured Programming*, London: Academic Press, s. 1-82.
[15] Dijkstra E.W. 1989: *In reply to comments*. EWD1058.
[16] Emerson E. A. 2008: *The beginning of model checking: A personal perspective*, [w:] DBLP:conf/spin/5000, s. 27-45.
[17] Giunchiglia F., Traverso P. 1999: *Planning as model checking*, [w:] *Proceedings of the Fifth European Workshop on Planning, (ECP'99)*, Springer, s. 1-20.
[18] Grumberg O., Veith H. (eds) 2008: *Years of Model Checking – History, Achievements, Perspectives*, (Lecture Notes in Computer Science, Vol. 5000), Springer.
[19] Hamblin C. L. 1970: *Fallacies*, London: Methuen.
[20] Hamblin C. L. 1987: *Imperatives*, Oxford: Basil Blackwell.
[21] Heath J., Kwiatowska M., Norman G., Parker D., Tymchysyn O. 2006: *Probabalistic model checking of complex biological pathways*, [w:] C. Priami (ed.), *Proc. Comp. Methods in Systems Biology, (CSMB'06)*, (Lecture Notes in Bioinformatics, Vol. 4210), Springer, s. 32-47.
[22] Hoare T. 2003: *The verifying compiler: A grand challenge for computing research*, „Journal of the ACM" vol. 50, 1, s. 63-69.
[23] Kaufmann M., Moore J.S. 2004: *Some key research problems in automated theorem proving for hardware and software verification*, „Revista de la Real Academia de Ciencias (RACSAM)". Serie A, 98 (1), s. 181-196.
[24] Kautz H., Selman B. 1992: *Planning as satisfiability*, [w:] *ECAI '92: Proceedings of the 10th European conference on Artificial intelligence*, New York, NY: John Wiley & Sons, s. 359–363.
[25] Künzel W. 2006: *The birth of the machine: Raymundus Lullus and his invention*, http://www.c3.hu/scca/butterfly/Kunzel/synopsis.html
[26] Leibniz G.W. 1685: *Machina arithmetica in qua non additio tantum et subtractio sed et multiplicatio nullo, divisio vero paene nullo animi labore peragantur*, Vol. Math. III A.2-c.
[27] Leibniz G.W. 1697: *Brief an den Herzog von Braunschweig-Wolfenbüttel Rudolph August*, 2. Januar 1697', http://www.fhaugsburg.de/ harsch/germanica/Chronologie/17Jh/ Leibniz/ lei_-bina.html
[28] Leibniz G.W. 1890: *Philosophische Schriften*, Vol. VII, Berlin.
[29] Leibniz G.W. 2006: *Sämtliche Schriften und Briefe*, (Zweite Reihe Philosophischer Briefwechsel, Vol. 1), Akademie Verlag.

[30] Manna Z., Pnueli A. 1992, 1995: *The Temporal Logic of Reactive and Concurrent Systems*, Vol. 1: *Specification*, 2: *Safety*, New York.
[31] Pnueli A. 1977: *The temporal logic of programs*, [w:] *Proceedings of the 18th IEEE-CS Symposium on Foundation of Computer Science (FOCS-77)*, IEEE Computer Society Press, s. 46-57.
[32] Pnueli A. 1981: *The temporal semantics of concurrent programs*, „Theoretical Computer Science" vol. 13, s. 45-60.
[33] Pratt V.R. 1980: *Applications of modal logic to programming*, „Studia Logica", s. 257-274.
[34] Prior A.N. 1967: *Past, Present and Future*, Oxford University Press.
[35] Prior A.N. 1996: *A statement of temporal realism*, [w:] B.J. Copeland (ed.), *Logic and Reality: Essays on the Legacy of Arthur Prior*, Oxford University Press.
[36] Queille J.-P., Sifakis J. 1982: *Specification and verification of concurrent systems in CESAR*, [w:] *Proceedings 5th International Symposium on Programming*, (Lecture Notes in Computer Science, Vol. 137), Springer-Verlag, s. 337-351.
[37] Quine W.V.O. 1966: *On the application of modern logic*, [w:] *The Ways of Paradox and Other Essays*, New York: Random House, s. 35–41.
[38] Rescher N. Urquhart A. 1971, *Temporal Logic*, Wien–New York: Springer.
[39] Schnoebelen P. 2002: *The complexity of temporal logic model checking*, „Advances in Modal Logic", s. 1-44.
[40] Trzęsicki K. 2005: *Arthura Normana Priora związki ze szkołą lwowsko-warszawską*, [w:] K. Trzęsicki (red.), *Ratione et studio*, Białystok: Uniwersytet w Białymstoku, s. 269-288.
[41] Trzęsicki K. 2006a: *From the idea of decidability to the number Ω*, „Studies in Grammar, Logic and Rethoric" (22), s. 73-142.
[42] Trzęsicki K. 2006b: *Leibniza idea systemu binarnego*, [w:] J. Kopania, H. Święczkowska (red.), *Filozofia i myśl społeczna XVII w.*, Białystok, s. 183-203.
[43] Trzęsicki K. 2006c: *Leibnizjańskie inspiracje informatyki*, „Filozofia Nauki" (3), s. 21-48.
[44] Turing A.M. 1936-1937: *On computable numbers, with an application to the Entscheidungsproblem*, „Proceedings of the London Mathematical Society" (Series 2), s. 230-265. Received May 25, 1936; Appendix dodany 28 sierpnia; read November 12, 1936; corrections ibid. vol. 43 (1937), s. 544-546. Artykuł Turinga ukazał się w Part 2 vol. 42, wydanym w grudniu 1936 (Przedruk w: M. Davis (ed.) 1965, s. 116-151; corr. ibid. pp. 151–154). Online: http://www.abelard.org/turpap2/tp2-ie.asp
[45] Turing A.M. 1986: *Lecture to the London Mathematical Society on 20 February 1947*, [w:] B. Carpenter, R. Doran (eds), *A. M. Turing's ACE Report of 1946 and Other Papers*, Cambridge, Mass.: MIT Press.
[46] Wang W., Hidvegi Z., Bailey A., Whinston A. 2000: *E-process design and assurance using model checking*, „IEEE Computer" (10), s. 48-53.

TEMPORAL LOGIC IN COMPUTER SCIENCE

Summary

Leibniz intended to create a *lingua characteristica universalis*, a language, in which all knowledge (expressiveness) could be recorded, and a calculus ratiocinator, a method that would make it possible to define in a calculational manner the truth of any sentence in this language. Such an idea is at the basis of contemporary logic. With respect to the correctness of the equipment and of programs such an idea is at the foundations of logical methods of verification of software systems.

Today we can see that a further development of informatics indeed depends on the progress of research in logic, and logic – this ancient discipline – has found a rich field for investigations; it has gained new perspectives for studies. Its applications may have a significant practical dimension that philosophers have never dreamed about.

Translated by Tadeusz Karłowicz

Słowa kluczowe: logika, informatyka, logika temporalna, weryfikowanie systemów informatycznych.

Key words: logic, computer science, temporal logic, verification of software systems.

Information about Author: KAZIMIERZ TRZĘSICKI, Ph.D. – Head of the Department of Logics, Informatics and Philosophy of Science, University of Białystok; address for correspondence: ul. Sosnowa 64, PL 15-887 Białystok, room 205; email: kasimir@ii.uwb.edu.pl