

<https://doi.org/10.25312/j.10411>

Anna Obrębska-Woźniczka  <https://orcid.org/0009-0005-3943-9461>
Uniwersytet Łódzki
e-mail: anna.obrebska@filologia.uni.lodz.pl

Programowanie jako praktyka komunikacyjna. Analiza języka programowania Pascal w perspektywie gramatyki komunikacyjnej

Programming as a communicative practice: An analysis of the Pascal programming language from the perspective of communicative grammar

Streszczenie

Artykuł przedstawia próbę zastosowania aparatu pojęciowego gramatyki komunikacyjnej do analizy języka programowania Pascal. Jego celem jest wstępne określenie przydatności trójpoziomowego modelu tej teorii w opisie języka programowania oraz wskazanie wybranych podobieństw i różnic między kodem a językiem naturalnym. Materiał stanowią reprezentatywne konstrukcje składniowe i semantyczne Pascala, poddane analizie jakościowej na poziomie ideacyjnym, interakcyjnym i organizacji dyskursu. Analiza wykazała, że na poziomie ideacyjnym podstawowym mechanizmem organizacji sensu jest układ predykatowo-argumentowy: funkcję predykatów pełnią przede wszystkim procedury, funkcje i niektóre operatory, natomiast rolę argumentów – zmienne, stałe i wyrażenia. Typy zmiennych wyznaczają status obiektów oraz zakres dopuszczalnych operacji. Na poziomie interakcyjnym instrukcje Pascala można interpretować jako sformalizowane odpowiedniki aktów dyrektywnych, kierowanych do maszyny i realizowanych bezwarunkowo albo zależnie od spełnienia warunków logicznych. Poziom organizacji dyskursu ujawnia się w elementach delimitujących i porządkujących kod, takich jak bloki instrukcji, sekcje deklaracyjne oraz definicje procedur i funkcji. Wyniki wskazują, że gramatyka komunikacyjna może być użytecznym narzędziem opisu Pascala jako szczególnej

formy komunikacji człowiek–maszyna, jednak możliwość zastosowania tego modelu do innych języków programowania wymaga dalszych analiz.

Słowa kluczowe: gramatyka komunikacyjna, komunikacja człowiek–maszyna, język programowania Pascal, akt mowy, funkcja impresyjna, komunikacja specjalistyczna, układ predykatowo-argumentowy

Abstract

This article explores the application of the conceptual framework of communicative grammar to the analysis of the Pascal programming language. Its aim is to provide a preliminary assessment of the usefulness of the theory's three-level model for describing a programming language and to identify selected similarities and differences between source code and natural language. The material comprises representative syntactic and semantic constructions in Pascal, subjected to qualitative analysis at the ideational, interactional, and discourse-organisation levels. The analysis demonstrates that, at the ideational level, meaning is organised primarily through predicate–argument structures: procedures, functions, and certain operators act as predicates, while variables, constants, and expressions function as arguments. Data types determine the status of entities and the range of permissible operations. At the interactional level, Pascal instructions can be interpreted as formalised counterparts of directive speech acts addressed to the machine and executed either unconditionally or upon the fulfilment of specified logical conditions. The discourse-organisation level is manifested in elements that delimit and structure code, including instruction blocks, declaration sections, and procedure and function definitions. The findings suggest that communicative grammar may provide a useful framework for describing Pascal as a specific form of human–machine communication. However, further research is required to establish whether this model can be applied to other programming languages.

Keywords: communicative grammar, human–machine communication, Pascal programming language, speech act, conative function, specialized communication, predicate-argument structure

Informatyka na tle innych dyscyplin naukowych pozostaje nadal młodą dziedziną wiedzy, a jej dynamiczny rozwój wiąże się między innymi z tym, że „nieustannie odkrywamy lepsze sposoby rozwiązywania problemów” (Scott, 2006: 5). Jednym z takich problemów wymagających uwagi jest choćby skuteczna i efektywna komunikacja ludzi i maszyn, ale także zespołów programistów, którzy współpracują nad złożonymi systemami, korzystając z rozmaitych języków programowania. Wobec niezwyklej akceleracji rozwoju technologii, w tym ekspansji zastosowań AI, „sytuacje, podczas których dochodzi do interakcji człowiek–maszyna, choć nie zawsze wydają się oczywiste, wypełniają codzienne życie jednostek” (Skrzypiec, 2024: 164), a „oprogramowanie jest wszechobecne we współczesnym społeczeństwie” (Allamanis i in., 2019: 1). W dzisiejszych czasach już niemal każdy aspekt życia, w tym energetyka, finanse, bezpieczeństwo, transport, rozrywka, zależy od „niezawodnego działania wysokiej jakości oprogramowania” (Allamanis i in., 2019: 1). Ten stan rzeczy powoduje, że niwelowanie barier językowych między człowiekiem a komputerem stało się „nieuniknionym kierunkiem rozwoju” (Rahit, Nabil, Huq, 2019: 1).

Aktualna dynamika rozwoju prac nad dużymi modelami językowymi (LLM) stawia nas wobec faktu, że najprawdopodobniej jesteśmy u początków nowej ery dystrybucji i przetwarzania informacji – ery AI, która ostatecznie zrewolucjonizuje wszystkie dziedziny życia, podobnie jak dokonały tego wcześniejsze przełomowe wynalazki. Bartosz Sawicki,

Michał Śmiałek i Bartłomiej Skowron (2024: 1) zauważają, że już „obecnie generatywne systemy sztucznej inteligencji potrafią tworzyć oprogramowanie na podstawie krótkich opisów sformułowanych w naturalnym języku ludzkim”. Przy czym praktyka dowodzi, że o ile AI nieźle sobie radzi z prostymi i typowymi zadaniami programistycznymi, o tyle jeszcze nie jest w stanie sprostać bardziej złożonym zadaniom programistycznym. Za celną można uznać uwagę Colina S. Gordona, który w artykule *The linguistics of programming* stwierdza, iż „wrażenie, że całe językoznawstwo zostało zdominowane przez LLM-y, jest podobne do przekonania laika, że programiści zasadniczo przestali programować i po prostu używają ChatGPT do generowania większości swojego kodu” (Gordon, 2024: 164). Zdaniem autora przekonanie to może wydawać się zrozumiałe w kontekście najpopularniejszych obecnie źródeł wiedzy, ale w rzeczywistości oparte jest „na stronniczych i wysoce niepełnych źródłach informacji” (Gordon, 2024: 164). Choć wydaje się, że znajomość języków programowania nadal będzie istotną kompetencją, badacze muszą zmierzyć się z pytaniem: „jak będzie wyglądało programowanie w erze silnej sztucznej inteligencji” (Sawicki, Śmiałek, Skowron, 2024: 2). Można przypuszczać, że języki programowania nowej generacji będą w większym stopniu dostosowane do ich użytkowników, albowiem:

W procesie kompilacji kod źródłowy programu jest tłumaczony na kod maszynowy dostosowany do architektury procesora, na którym jest wykonywany. Jednocześnie służy on jako kluczowe narzędzie komunikacji między programistami. Umiejętność zrozumienia, kontrolowania i korygowania sposobu działania programu buduje zaufanie do programu komputerowego i ułatwia współpracę (Sawicki, Śmiałek, Skowron, 2024: 2).

Specyfika języków programowania

Według dostępnych źródeł „istnieje ponad tysiąc aktywnie utrzymywanych języków programowania, co wskazuje na ich dużą różnorodność. Przy czym języki te zostały stworzone do realizacji różnych celów i wykorzystują odmienne składnie” (Rahit, Nabil, Huq, 2019: 2), a na informatycznych studiach pierwszego stopnia „nierzadko można spotkać użytkowników wielu różnych języków” (Scott, 2006: 5).

Jak pisze Ricardo Maciel Gazoni (2018: 97), „języki programowania są zazwyczaj przedstawiane jako mechanizmy upraszczające korzystanie z jednostki centralnej komputera (CPU). Urządzenia te działają w oparciu o ładunki elektryczne, które można rozumieć jako bity”. Sekwencje bitów można z kolei postrzegać jako ciąg cyfr będących w rzeczywistości instrukcjami do wykonania. Język programowania pozwala wydawać komputerowi te instrukcje za pomocą prostych, bazujących na języku angielskim komunikatów, które system komputera przetwarza na ciąg cyfr w systemie zerojedynekowym. Wyróżnia się języki programowania wysokiego oraz niskiego poziomu – im bardziej język programowania jest „przyjazny dla użytkownika”, a więc bliski systemowi języka naturalnego, tym wyższy jest jego poziom. W odpowiedzi na potrzebę ułatwienia komunikacji programisty z maszyną już w połowie XX stulecia powstał pierwszy język wysokiego poziomu, którym

był – obecnie prawie nieużywany – Fortran. Na długo jednym z najpopularniejszych, uniwersalnych języków programowania wysokiego poziomu stał się z kolei opracowany około 1970 roku Pascal, który uważany jest za język „dydaktyczny”, więc nadal bardzo często właśnie od niego informatycy rozpoczynają naukę programowania.

Z punktu widzenia lingwistyki prawdopodobnie jedną z najciekawszych właściwości języków programowania jest to, że kod źródłowy programu ma dwa zupełnie różne typy odbiorców, ponieważ służy zarówno komunikacji ludzi z maszynami, jak i ludzi z innymi ludźmi. Jak zauważają autorzy artykułu *A survey of machine learning for big code and naturalness*, „ludzie muszą rozumieć kod, aby móc go czytać i pisać; komputery muszą być w stanie go wykonywać. Ta bimodalność kodu determinuje zarówno podobieństwa, jak i różnice między nim a tekstem” (Allamanis i in., 2019: 4).

Istotnym wyróżnikiem języków programowania jest to, że „każdy język programowania konstruowany jest w taki sposób, aby nie dopuszczał niejednoznacznych słów [...]. Wszystko, co zostaje zapisane w języku programowania, musi zostać przetłumaczone na instrukcje CPU, które ostatecznie pobudzą obwody elektroniczne w ściśle określony sposób” (Gazoni, 2018: 97). Inaczej niż w językach naturalnych języki programowania wyrażają „jedynie to, co może zostać dalej przetłumaczone na instrukcje CPU” (Gazoni, 2018: 98).

Najistotniejszą różnicą między językami naturalnymi i językami programowania jest jednak to, że te drugie reprezentują języki formalne, które są projektowane sztucznie. W związku z tym są stwarzane arbitralnie przez nielicznych projektantów. Jak słusznie zauważa Miltiadis Allamanis ze współautorami, „języki naturalne zmieniają się stopniowo, podczas gdy języki programowania doświadczają zmian skokowych: nowe wydania, takie jak Python 3, czasami zrywają wsteczną kompatybilność” (Allamanis i in., 2019: 5). Ponadto język programowania wykazuje niewielką lub żadną elastyczność na odstępstwa od wzorcowej normy, na przykład „formatowanie może mieć w kodzie znaczenie – klasycznym przykładem jest wrażliwość Pythona na białe znaki” (Allamanis i in., 2019: 5). W konsekwencji najdrobniejszy błąd w zapisie zazwyczaj czyni komunikat zupełnie niezrozumiałym dla komputera. Również autorzy artykułu *Studying the difference between natural and programming* podkreślają, że „w kodzie koszt błędnej interpretacji jest bardzo wysoki, dlatego gramatyka zazwyczaj nie dopuszcza niejednoznaczności”, zapewne dlatego jest on „od około 8 do 16 razy bardziej przewidywalny” (Casalnuovo, Sagae, Devanbu, 2018: 32) niż język naturalny, który „obfituje w niejednoznaczności gramatyczne” (Casalnuovo, Sagae, Devanbu, 2018: 5).

Niezależnie od wszystkich fundamentalnych różnic badacze zwracają także uwagę na liczne podobieństwa języków naturalnych i języków programowania. Na przykład K.M. Tahsin Hassa Rahit, Rashidul Nabil i Md. Hasibul Huq zauważają, że „język programowania jest w pewnym sensie podobny do języka naturalnego, lecz dysponuje znacznie uboższym słownictwem w porównaniu z typowym językiem ludzkim” (Rahit, Nabil, Huq, 2019: 4). Co ciekawe, dostrzega się też podobieństwa w sposobie ewolucji języków naturalnych i języków programowania. Na przykład Colin S. Gordon zauważa, że „dwa główne sposoby, w jakie ewoluują języki programowania, to zapożyczanie cech

z innych języków oraz tworzenie skrótów służących do wyrażania często używanych idiomów, które pierwotnie były bardziej rozbudowane”, i – jak twierdzi badacz – jest to też typowe dla ewolucji języków naturalnych (Gordon, 2024: 171).

Rozpatrując podobieństwa pomiędzy językami naturalnymi i językami programowania, nie sposób pominąć faktu, że większość języków programowania wprost bazuje na znajomości angielskiego, z którego zapożyczane są na przykład nazwy funkcji w kodzie programu. To czerpanie nazewnictwa z języka angielskiego jest rozwiązaniem niezwykle praktycznym, bowiem – jak twierdzą autorki artykułu *The language of programming: a cognitive perspective*, „nieintuicyjne” nazewnictwo zwiększa obciążenie poznawcze oraz utrudnia rozumienie programu (Fedorenko i in., 2019: 4). Badaczki zauważają ponadto, że kod programu jest często dopełniany dodatkową dokumentacją lub komentarzami, które pełnią funkcję „rusztowania poznawczego wspomagającego zrozumienie programu”. Także i to powoduje, że „praca z kodem nieuchronnie wiąże się ze ścisłą integracją wiedzy z zakresu języków komputerowych i naturalnych” (Fedorenko i in., 2019: 4).

Programowanie jako praktyka komunikacyjna

Hipoteza naturalności, którą postawiono w publikacji *A survey of machine learning for big code and naturalness*, zakłada, że „oprogramowanie jest formą komunikacji międzyludzkiej; korpusy oprogramowania mają podobne własności statystyczne jak korpusy języka naturalnego; a własności te można wykorzystać do budowy lepszych narzędzi inżynierii oprogramowania” (Allamanis i in., 2019: 3). Także autorzy artykułu *On the naturalness of software* podkreślają, że programowanie jest „w równym stopniu aktem komunikacji między ludźmi oraz sposobem mówienia komputerom, co mają robić” (Hindle i in., 2016: 122). Z tezą, że posługiwanie się językiem programowania stanowi formę komunikacji, zgadzają się również Evelina Fedorenko, Anna Ivanova, Riva Dhamala i Marina Umaschi Bers, przy czym badaczki zwracają uwagę na fakt, że „założenia właściwe komunikacji w języku naturalnym nie mają zastosowania w programowaniu” (Fedorenko i in., 2019: 8). Niedostrzeżenie tej różnicy skutkuje błędami związanymi z niezamierzonym przenoszeniem założeń dotyczących wiedzy powszechnej i trywialnej na komunikację człowieka z maszyną (Fedorenko i in., 2019: 8).

Chociaż za pierwotną rolę języków programowania uznaje się „pełnienie funkcji medium komunikacji między człowiekiem a komputerem” (Dubochet, 2009: 1), to jednak współcześnie w coraz większym stopniu dostrzega się, że „jednocześnie język programowania odgrywa kluczową rolę jako narzędzie komunikacji między programistami” (Sawicki, Śmiałek, Skowron, 2024: 2). Jest to szczególnie istotne w czasach, gdy „cykl życia oprogramowania uległ wydłużeniu, a zespoły programistyczne znacznie się powiększyły” (Dubochet, 2009: 1). Nie bez znaczenia jest też to, że „programiści preferują pisać i czytać kod konwencjonalny, idiomatyczny i znajomy, ponieważ ułatwia to zrozumienie oraz utrzymanie systemów informatycznych” (Allamanis i in., 2019: 4). Języki programowania mogą być zatem interesującym obiektem badań zarówno w kontekście komunikacji język–maszyna, jak i jako narzędzie komunikacji specjalistycznej w środowisku programistów.

Język programowania Pascal w ujęciu gramatyki komunikacyjnej

Zdaniem Colina S. Gordona „lingwistyka oferuje bogaty zasób idei, perspektyw oraz ugruntowanych narzędzi analitycznych (zarówno formalnych, jak i nieformalnych), które [...] mogą pomóc w lepszym zrozumieniu problemów istotnych dla badaczy języków programowania oraz inżynierii oprogramowania” (Gordon, 2024: 162). Ponadto zwraca się uwagę na fakt, że „społeczność zajmująca się uczeniem maszynowym już zaczęła wykorzystywać strukturalne podobieństwa między kodem a językiem naturalnym, stosując techniki przetwarzania języka naturalnego (NLP) do analizy kodu” (Fedorenko i in., 2019: 4). Jedną z ciekawych perspektyw lingwistycznych, którą można zastosować do analizy języków programowania, jest gramatyka komunikacyjna, należąca do szeroko rozumianego nurtu gramatyk funkcjonalnych. Czołowi przedstawiciele i twórcy gramatyki komunikacyjnej, Aleksy Awdiejew i Grażyna Habrajska, przyjmują za Michaeliem A.K. Hallidayem, że procesy komunikacyjne „przebiegają na trzech funkcjonalnie różnych poziomach gramatyki” (Awdiejew, Habrajska, 2004: 34): ideacyjnym, interakcyjnym i organizacji dyskursu. W dalszej części artykułu klasyczny język programowania Pascal zostanie poddany analizie z perspektywy tych trzech poziomów funkcjonowania tekstu. Celem tego zestawienia nie jest jedynie wskazanie różnic pomiędzy językami naturalnymi a językami programowania, lecz także wstępne sprawdzenie zakresu użyteczności aparatu pojęciowego gramatyki komunikacyjnej w opisie języków programowania.

Poziom ideacyjny

Jak podają Awdiejew i Habrajska, „najważniejszym poziomem funkcjonalnym języka, jaki się przejawia w większości jego realizacji [...] jest zdaniem Hallidaya poziom ideacyjny (przedstawieniowy), którego podstawowym celem jest [...] obrazowanie świata” (Awdiejew, Habrajska 2004: 34). Ten poziom jest w językach programowania reprezentowany znacznie słabiej niż w językach naturalnych, co wynika z ograniczonych funkcji tych pierwszych. Mimo iż najsilniej wyróżnia się w języku Pascal funkcja impresywna, poziom interakcyjny nie jest w stanie istnieć bez odniesienia ideacyjnego, więc na przykład polecenia *write* czy *read* mają także swój wymiar przedstawieniowy. W języku naturalnym podstawowym nośnikiem obrazów ideacyjnych jest układ predykatowo-argumentowy, którego konstytutywnym elementem jest predykat „otwierający miejsce dla argumentów (nazw obiektów, przedmiotów, osób, zjawisk itd.)” (Awdiejew, Habrajska, 2004: 36).

W języku Pascal miejsce predykatów wypełniają nazwy procedur, funkcji oraz niektóre operatory, natomiast rolę argumentów realizują zmienne, stałe oraz wyrażenia. Predykaty proceduralne, takie jak *writeln* czy *readln*, inicjują określone działania programu, przyjmując jako argumenty dane, na których operacje są wykonywane. Na przykład w instrukcji: *readln(imie)* procedura *readln* pełni funkcję predykatu, zaś pozycję argumentu zajmuje zmienna *imie*, w której zostaje zapisana wartość wprowadzona przez użytkownika. Analogicznie w instrukcji *writeln('Witaj świecie')* predykat *writeln* przyjmuje jako argument stałą znakową. Również funkcje takie jak *sqrt* realizują strukturę predykatowo-argumentową,

przy czym rezultatem ich działania jest wartość, która zostaje przypisana zmiennej. Na przykład w instrukcji $x := \text{sqrt}(16)$ obliczająca pierwiastek kwadratowy funkcja *sqrt* odgrywa rolę predykatu, który odzwierciedla relację matematyczną pomiędzy argumentem a wynikiem. Na poziomie ideacyjnym sytuować też należy operatory relacyjne, które orzekają na temat zależności pomiędzy argumentami. Na przykład jeśli procedurze nadamy warunek: *if x > 0 then writeln('Liczba dodatnia')*, znak „>”, oznaczający *jest większe*, spełniać będzie funkcję predykatu określającego relację pomiędzy argumentami x i 0. Wynikiem tej predykcji będzie tu pewna wartość logiczna determinująca dalsze działanie programu.

Miejsce argumentów w układach predykatowo-argumentowych mogą wypełniać nie tylko pojedyncze zmienne lub stałe, lecz także złożone wyrażenia, na przykład *writeln(a + b * c)*. W tej sytuacji metapredykat *writeln* przyjmuje jako argument całe wyrażenie arytmetyczne, w którym operatory $+$ i $*$ tworzą zagnieżdżoną strukturę predykacyjną. Mechanizm ten odpowiada złożonym konstrukcjom predykatywowym znanym z języków naturalnych.

Do poziomu ideacyjnego należy również operator negacji *not*, który modyfikuje wartość logiczną wyrażenia. Ponadto na poziomie ideacyjnym usytuować możemy typy zmiennych, takie jak *integer*, *char* czy *boolean*. Należą one do poziomu przedstawieniowego, gdyż określają ontologiczny status obiektów świata programu. Nie realizują one same w sobie predykcji, ale wyznaczają ramy semantyczne, w obrębie których predykaty proceduralne, funkcyjne czy relacyjne mogą się z danymi argumentami łączyć. Poziom ideacyjny języka reprezentują także często używane w językach programowania operatory logiczne, na przykład *and*, *or*, *xor*, które konceptualizują zależności logiczne między stanami rzeczy, uczestnicząc w ten sposób w konstruowaniu treści przedstawieniowej. Operator *or* znajduje zastosowanie, gdy programista chce zaznaczyć, że przynajmniej jeden z minimum dwóch warunków musi być spełniony, aby zaszło określone zdarzenie. W przypadku operatora logicznego *xor* instrukcja wykona się tylko wtedy, gdy dokładnie jeden z dwóch warunków zostanie spełniony. Nie można zapomnieć również o bardzo istotnym operatorze *and*, który powoduje, że polecenie zostanie wykonane, jeśli wszystkie zadeklarowane warunki zostaną spełnione.

Poziom interakcyjny

Jak podają Awdiejew i Habrajska (2024: 38), „na poziomie interakcyjnym nadawca w sposób subiektywny realizuje swój zamiar pragmatyczny”, a „podstawową jednostką opisu na tym poziomie jest akt mowy”, dzięki któremu komunikat spełnia funkcje pragmatyczne.

W języku Pascal dominuje funkcja impresyjna, charakterystyczna dla poziomu interakcyjnego języka, która rozumiana jest tu jako formalny odpowiednik aktów dyrektywnych znanych z języków naturalnych. Realizuje się ona poprzez instrukcje i procedury nakazujące komputerowi wykonanie określonych poleceń. Jako przykład posłużyć mogą procedury wejścia i wyjścia, które reprezentowane są poprzez zaczerpnięte z języka angielskiego słowa *read* czy *write* oraz wyrażenia pokrewne *writeln* i *readln* (Przykład 1).

Przykład 1

```
writeln('Podaj liczbę');  
readln(x);
```

Procedury tego rodzaju odpowiadają funkcjonalnie aktom żądania, przy czym w języku programowania realizacja tych aktów ma charakter bezwarunkowy i nienegocjowalny.

Do poziomu interakcyjnego języka można również zaliczyć instrukcje sterujące przebiegiem programu, takie jak pętle czy instrukcje warunkowe. Na przykład pętla *repeat... until...* realizuje blok poleceń aż do momentu otrzymania założonego rezultatu (*Przykład 2*).

Przykład 2

```
repeat  
  writeln('Podaj hasło');  
  readln(hasło);  
until hasło = 123;
```

Odmienne działa instrukcja *while... do...*, w której warunek kontrolujący wykonywanie poleceń sprawdzany jest na początku (*Przykład 3*):

Przykład 3

```
while licznik < 10 do  
begin  
  writeln(licznik);  
  licznik := licznik + 1;  
end;
```

Nieco bardziej rozbudowaną składnię ma instrukcja warunkowa *if... then... else*, uruchamiająca jedno z alternatywnych działań programu w zależności od spełnienia określonego warunku (*Przykład 4*).

Przykład 4

```
if x >= 0 then  
  writeln('Liczba nieujemna')  
else  
  writeln('Liczba ujemna');
```

Wszystkie te konstrukcje pełnią funkcję dyrektywną i są odpowiednikami aktów mowy ukierunkowanych na wywołanie określonego efektu.

W przeciwieństwie do języka naturalnego w języku programowania na poziomie interakcyjnym języka zupełnie nie występują akty modalne, akty emotywno-oceniające czy perswazyjne. Język Pascal realizuje wyłącznie akty dyrektywne, przy czym mają one charakter skierowanych do maszyny poleceń, które mogą przyjmować postać: instrukcji

bezwzględnie inicjujących działanie; instrukcji, których realizacja zależy od uprzedniego spełnienia określonego warunku, a także instrukcji nakazujących wielokrotne wykonanie działania.

Poziom organizacji dyskursu

Funkcjonujące na poziomie organizacji dyskursu operatory porządkujące przekaz informacyjny „w odróżnieniu od operatorów interakcyjnych nie wprowadzają jakichkolwiek aktów mowy, nie zmieniają samej wartości interakcji, lecz w różny sposób przedstawiają tę samą treść lub usprawniają proces odbioru treści na poziomie metatekstowym” (Awdiejew, Habrajska, 2004: 39). Do tego poziomu można w języku Pascal zaliczyć przede wszystkim elementy delimitujące i segmentujące kod, takie jak wyrażenia *begin* i *end*, które wyznaczają granice bloków instrukcji przeznaczonych do wykonania. Na poziomie organizacji dyskursu sytuują się również elementy wprowadzające i porządkujące strukturę programu, takie jak sekcje deklaracyjne oznaczane słowem *var*, a także definicje procedur i funkcji. Pełnią one funkcję metatekstową, wskazując sposób organizacji kodu oraz relacje między jego poszczególnymi częściami. Elementy te służą strukturyzacji tekstu programu, wzmacniając jego czytelność oraz umożliwiając właściwą interpretację zarówno maszynom, jak i ludziom.

Podsumowanie

Przeprowadzona analiza ukazuje, że języki programowania – mimo swojej sztucznej natury – mogą być z powodzeniem opisywane przy użyciu metodologii gramatyki komunikacyjnej. Odwołanie się do trójpoziomowego modelu opisu języka pozwoliło uchwycić zarówno elementarne podobieństwa, jak i zasadnicze różnice między językami naturalnymi a językami programowania. Na poziomie ideacyjnym funkcję predykatów pełnią przede wszystkim procedury i funkcje, natomiast rolę argumentów – zmienne, stałe i wyrażenia. Badania wykazały, że również w języku programowania sens przedstawieniowy konstituuje się poprzez relacje predykacyjne, w tym także struktury zagnieżdżone. Istotną rolę odgrywają tu też typy zmiennych, które określają ontologiczny status obiektów, oraz operatory logiczne, organizujące zależności między predykacjami. Analiza poziomu interakcyjnego wykazała, że w języku Pascal dominują akty dyrektywne, realizowane w postaci kierowanych do maszyny jednoznacznych poleceń, które pozbawione są elementów modalnych, ekspresywnych czy perswazyjnych. Komunikacja ma charakter jednostronny, a realizowane akty dyrektywne przyjmują postać formalnie określonych poleceń, których wykonanie bywa uzależnione od warunków logicznych. Funkcja impresywna przyjmuje zatem w języku Pascal postać silnie zredukowaną, podporządkowaną wykonaniu instrukcji, co stanowi jedną z najbardziej wyrazistych cech odróżniających języki programowania od języków naturalnych. Poziom organizacji dyskursu ujawnia się natomiast w elementach strukturyzujących kod, takich jak wyrażenia *begin* i *end* czy sekcja deklaracji zmiennych. Pełnią one funkcję metatekstową, porządkując kod programu

i umożliwiając jego właściwą interpretację. Nie wprowadzają przy tym nowych treści ideacyjnych ani aktów mowy.

Przeprowadzone analizy potwierdziły, że gramatyka komunikacyjna może stanowić użyteczne narzędzie opisu języków programowania. Ponadto pozwoliły nie tylko lepiej uchwycić specyfikę komunikacji człowiek–maszyna, lecz także ukazać języki programowania jako szczególny przypadek komunikacji specjalistycznej. Analiza lingwistyczna może stanowić istotne uzupełnienie badań nad naturą programowania oraz nad jego funkcjonowaniem w coraz bardziej złożonym świecie współczesnych technologii.

Bibliografia

Allamanis M., Barr E.T., Devanbu P., Sutton C. (2019), *A survey of machine learning for big code and naturalness*, „ACM Computing Surveys”, vol. 51(4), s. 1–37, <https://doi.org/10.1145/3212695>

Awdziejew A., Habrajska G. (2004), *Wprowadzenie do gramatyki komunikacyjnej*, t. 1, Łask: Oficyna Wydawnicza Leksem.

Casalnuovo C., Sagae K., Devanbu P. (2018), *Studying the difference between natural and programming language corpora*, „Empirical Software Engineering”, vol. 24, s. 1823–1868, <https://doi.org/10.48550/arXiv.1806.02437>

Dubochet G. (2009), *Computer code as a medium for human communication: Are programming languages improving?*, [w:] *Proceedings of the 21st Annual Workshop of the Psychology of Programming Interest Group (PPIG)*, Limerick: University of Limerick.

Fedorenko E., Ivanova A., Dhamala R., Bers M.U. (2019), *The language of programming: A cognitive perspective*, „Trends in Cognitive Sciences”, vol. 23(7), s. 525–528, <https://doi.org/10.1016/j.tics.2019.04.010>

Gazoni R.M. (2018), *A semiotic analysis of programming languages*, „Journal of Computer and Communications”, vol. 6, s. 91–101, <https://doi.org/10.4236/jcc.2018.63007>

Gordon C.S. (2024), *The linguistics of programming*, [w:] *Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '24)*, New York: ACM, s. 162–182, <https://doi.org/10.1145/3689492.3689806>

Hindle A., Barr E.T., Gabel M., Su Z., Devanbu P. (2016), *On the naturalness of software*, „Communications of the ACM”, vol. 59(5), s. 122–131, <https://doi.org/10.1145/2902362>

Rahit K.M.T.H., Nabil R.H., Huq M.H. (2019), *Machine translation from natural language to code using Long-Short Term Memory*, [w:] *Proceedings of the Future Technologies Conference (FTC) 2019. Advances in Intelligent Systems and Computing*, vol. 1069, Cham: Springer, s. 56–63, <https://doi.org/10.48550/arXiv.1910.11471>

Sawicki B., Śmiałek M., Skowron B. (2024), *Towards the ultimate programming language: Trust and benevolence in the age of artificial intelligence*, arXiv:2412.00206, <https://doi.org/10.48550/arXiv.2412.00206>

Scott M.L. (2006), *Programming language pragmatics*, San Francisco: Elsevier.

Skrzypiec A. (2024), *Plaszczyzny interakcji między człowiekiem a maszyną: Zarys problematyki z perspektywy nauki o komunikacji społecznej*, „Człowiek i Społeczeństwo”, t. 58, s. 149–164.