LingBaW

Linguistics Beyond And Within 5 (2019), 7-16

Explaining IT programming concepts using NSM explications: The case of 'variable' and 'constant'

Bartłomiej Biegajło
Siedlce University of Natural Sciences and Humanities, Poland

Abstract
The paper seeks to explore a practical application of Natural Semantic Metalanguage in defining two core concepts in computer programming, i.e. the concept of a variable and the concept of a constant. The investigation of both programming concepts is carried out with reference to Apple's Swift programming language, which is now the dominant language in creating applications designed for Apple's devices. The explications of a variable and a constant developed in this paper are tentative definitions of the most fundamental functionalities behind the two programming concepts. They are meant to ease the learning experience of programming enthusiasts who are at the early stages of their learning path.

Keywords: Natural Semantic Metalanguage, Swift programming language, explications, variable, constant

1. Introduction

As everyday users of a number of electronic devices, we are in constant need of passing various types of electronic data between a plethora of electronic machines. Vast amounts of data travel among users of mobile phones, tablets, laptop computers, desktop machines as they communicate with servers distributed around the world. Users can manipulate data – change their contents, add new items, delete unnecessary parts, or remove them altogether and, essentially, store them in memory of a device. Data are crucially virtual in form, which means that they are not tangible objects that can be physically experienced by touching them or feeling them. However, at the same time, data require some physical location to store the very contents these data contain.

We, the users, or participants of this continuous exchange of digital information are frequently unaware of the complex, yet very well defined rules that govern the flow of data. It is for the convenience of an end-user that the architecture of modern software is designed in such a way as to make the so-called user-experience (UX) pleasant and intuitive. Apple's online documentation concerning *Human Interface Guidelines* is a clear indication of what the company puts to the front of its UX:

> 'The best apps find the correct balance between enabling users and avoiding unwanted outcomes. An app can make people feel like they're in control by keeping interactive elements familiar and predictable, confirming destructive actions, and making it easy to cancel operations, even if they're already underway.'
>
> (Apple *Human Interface Guidelines*)

Apple's attitude towards the visual components of apps is not exclusive. In her bestselling book, *Practical UI Patterns for Design Systems*, Diana MacDonald reiterates a conclusion which echoes Apple's guidelines but entails a broader range of online user experience: "UI patterns (user interface patterns) are found in the digital sphere of websites, applications, native mobile apps, and other software or devices" (2019: 2). She also contends that "the small, reusable UI solutions found in these patterns can then be composed together to build cohesive, intuitive experiences that resonate with people" (2019: 11). In a nutshell, this is a win-win situation as these solutions are both familiar to the users and constitute a reusable source to app developers.

2. Under the hood

The visual layer of apps underlies user's digital experience, and for the vast majority of people, this is the sole quality to assess in judging whether an application is useful or not. However, UI constitutes just a front side of an app – it is simply an intuitive navigation tool which allows the manipulation of different types of data. The machinery that governs the exact routes taken by these data is a line of code written in one of the programming languages distributed free of charge and widely available to programmers around the world. In a most basic scenario, a skilled programmer writes a set of instructions which are then interpreted by a machine which in turn performs a set of actions based on the available instructions. The difficulty in developing such instructions lies in the fact that they should account for all the possible actions taken by a user, including the ones which, more often than not, can hardly be predicted. These skills come with experience, and this type of programming activity is known as backend programming. In other words, backend programmers are responsible for connecting the front side of an app (hence: frontend developers) with a set of instructions that can further be sent to an interpreting machine for reading and responding (hence: backend developers) and, eventually, sending back any data that are requested by a user working on his/her personal device.

There are different programming languages which developers use to create a meaningful user experience. A predefined set of syntactic rules must, by necessity, govern all those languages. Violating any of those rules leads to what developers call 'a crash of an app'. Although programming languages are examples of artificial languages, their elementary structures bear clear resemblance to how natural languages are organised into a set of meaningful units. Their syntax and semantics (keywords) and the specific manner in which the two must be combined into readable instructions in order for the interpreter to 'comprehend' them are the essence programming. Likewise, as is the case with any natural language, bad grammar results in bad communication.

The optimistic part of programming is that all of the commercial programming languages employ either the same or just slightly differing concepts which developers reuse to organise data flow. These concepts might include variables, constants, strings, numbers, Booleans,

arrays, dictionaries, sets, loops, functions, object-oriented programming and many more essential keywords that might well sound utterly foreign to a non-programmer. To make a life of a developer easier, all of these concepts are assigned specific predefined keywords which an interpreting machine can recognise and read in real-time. Learning basic programming concepts can take some time; however, it is possible to understand the essential syntax in a matter of one/two weeks as Bruce Tate indicates in his somewhat very optimistic book title *Seven Languages in Seven Weeks. A Pragmatic Guide to Learning Programming Languages* (2010).

The less optimistic part of programming, however, is turning these rudimentary concepts into a combination of meaningful expressions where, say, an array is in a dialogue with a function, or a class is supposed to fetch data from a function and assign them to a variable that eventually supplies a dictionary for later use. A parallel experience happens when one learns a natural language – although, after some time and effort, one can understand individual words, or even basic sentences, it is still extremely tough to maintain a fluent conversation in that language. Tate compares learning to programme with learning how to swim: "no amount of theory is a substitute for diving into the pool and flailing around in the water gasping for air" (2010: 14). Practice is the keyword for any learning path. While it is true that, especially in the case of learning how to programme digital applications, theory amounts to a convenient springboard for practice, novice programmers cannot ignore getting acquainted with basic programming concepts if they intend to start a fully-fledged career in the industry.

The following discussion will therefore invariably have to touch upon theoretical underpinnings of the two most basic concepts in programming, i.e. the concept of a variable and the concept of a constant. The programming language which this overview will be referring to is the Apple's flagship Swift programming language; however, the mechanism of using the concepts of a variable and a constant can be found in any programming language accessible in the public domain. Apple's rich and meticulous documentation secures a convenient reference source and, although full documentation is freely available on their website, in order to make navigation through Apple's numerous digital references easier, the following study will be referring to Apple's latest e-book on Swift language released by their programming community.

3. Theoretical background

The programming concepts of a variable and a constant are the most basic and most easy-to-grasp means for storing data. According to Apple's guide, *The Swift Programming Language*, "Swift uses variables to store and refer to values by an identifying name. Swift also makes extensive use of variables whose values can't be changed. These are known as constants" (2019: 49). As simple as the explanation apparently sounds, it merits a question of whether there is a need for any more descriptive words to be supplemented in an attempt to turn this definition into an even more exhaustive example of unambiguous meaning.

The primary focus of this paper is to apply the theoretical framework of Natural Semantic Metalanguage to develop explications of two programming concepts and to test them against a largely unexplored domain of linguistic enquiry, possibly helping programming newcomers

arrange their programming knowledge into a transparent and accessible reference book/e-book/app. Due to reasons of space, I am not going to provide a detailed overview of the methodology of Natural Semantic Metalanguage (for an up-to-date, almost handbook-like account of the nuts and bolts of NSM, see Goddard 2008, 2018a; Goddard and Wierzbicka 2014). However, a few introductory remarks are essential to grant this paper a solid theoretical foothold.

Natural Semantic Metalanguage has a history of research spanning more than 40 years. It was formally devised by Anna Wierzbicka who continually credits Andrzej Bogusławski to be her major inspiration: "my own interest in the pursuit of non-arbitrary semantic primitives was triggered by a lecture on this subject given at Warsaw University by the Polish linguist Andrzej Bogusławski in 1965" (1996: 13). Today, as Bogusławski admits, "there is a real school of semantics (with the label *NSM* – coming from 'natural semantic metalanguage') led by Wierzbicka (the 'second-in-command' is Cliff Goddard), […] its site has been, for thirty years now, in Australia" (2011: 80). In the simplest possible explanation, NSM is "the conceptual shared core of all languages [which] can serve as a neutral metalanguage for describing and comparing languages and all culture-specific discourses" (Wierzbicka 2013: 2). Elsewhere, Wierzbicka complements the above by stating: "the concomitant claims is, of course, that every word (other than the members of the basic 'alphabet') can be defined" (1992: 22). The very term in parenthesis, 'the basic alphabet' which Wierzbicka refers to, is a paraphrase of Leibniz's catchy expression – 'the alphabet of human thoughts': "Leibniz believed that every human being is born with a set of innate ideas which become activated and developed by experience but which latently exists in our minds from the beginning" (1992: 8). The 'innate ideas' seen from the perspective of Natural Semantic Metalanguage are part and parcel of human nature, they are therefore at the disposal of all human beings, regardless of the cultural background, language and geographical location. Wierzbicka further declares that "all complex thoughts – all meanings – arise through different combinations of simple ideas" (1992: 8) which NSM labels as semantic primes: "the elements which can be used to define the meaning of words (or any other meanings) cannot be defined themselves; rather, they must be accepted as 'indefinibilia', that is, as semantic primes, in terms of which all complex meanings can be coherently represented" (Wierzbicka 1997: 25). Goddard illustrates this assumption adding that "they [i.e. semantic primes] are simple and intuitively intelligible meanings grounded in ordinary linguistic experience" (2010: 462) which suggests that semantic primes are translatable cross-culturally as opposed to more complex words which are outside the bounds of NSM.

The extensive research carried out within NSM allowed NSM scholars to posit 65 semantic primes (also called 'semantic universals', or 'semantic primitives' to denote the same concept in a somewhat more powerful manner) which the scholars believe to constitute "the metalanguage of semantic representation [which] may be viewed as the smallest 'mini-language' with the same expressive power as the full natural language" (Goddard 1994: 12). It is simply a tool made out of any natural language (the assumption being that semantic primes connote exactly the same meaning, regardless of the language they are translated into) which is used to talk about less simple concepts found in those languages. Below is a complete list of 65 English semantic universals, otherwise known as non-definables, which the NSM scholars use to describe words,

expressions and concepts, i.e. definables, whose premise, according to NSM, is outside the category of primes.

I, YOU, SOMEONE, SOMETHING~THING, PEOPLE, BODY	Substantives
KIND, PART~HAVE PARTS	Relational substantives
THIS, THE SAME, OTHER~ELSE	Determiners
ONE, TWO, SOME, ALL, MUCH~MANY, LITTLE~FEW	Quantifiers
GOOD, BAD	Evaluators
BIG, SMALL	Descriptors
KNOW, THINK, WANT, DON'T WANT, FEEL, SEE, HEAR	Mental predicates
SAY, WORDS, TRUE	Speech
DO, HAPPEN, MOVE	Actions, events, movement
BE (SOMEWHERE), THERE IS, BE (SOMEONE/SOMETHING)	Location, existence, specification
(IS) MINE	Possession
LIVE, DIE	Life and death
TIME~WHEN, NOW, BEFORE, AFTER, A LONG TIME, A SHORT TIME, FOR SOME TIME, MOMENT	Time
PLACE~WHERE, HERE, ABOVE, BELOW, FAR, NEAR, SIDE, INSIDE, TOUCH	Space
NOT, MAYBE, CAN, BECAUSE, IF	Logical concepts
VERY, MORE	Augmentor, intensifier
LIKE	Similarity

Goddard (2018b: 63)

This very sketchy overview of Natural Semantic Metalanguage certainly neither accounts for the potential of the theory, nor for its inherent pitfalls. One such pitfall can be identified when one intends to examine the tenets of NSM from the point of view of translation studies. Research suggests that it is not a single concept/word that matters the most, but a whole, unified text is seen to determine the interpretation (which again, might be highly subjective) of a message/image/the type of imagination a given writer puts forward. Support for these reservations can be found particularly in functional theories of translation (Reiss 2000, 2004; Reiss and Vermeer 2013; Snell-Hornby 1995; Vermeer 2012). A recent direct critical assessment of NSM, addressing Wierzbicka's framework from the viewpoint of translation studies is Blumczyński (2013) who, among other things, stresses the importance of context in translation and the resulting unavoidable differences in individual interpretation of texts, including interpretation of separate words and/or concepts.

On the other hand, extensive literature exists whose primary aim is to further investigate and test NSM in different cultural contexts. Wierzbicka (2014: 247) supplements a selection of languages studied in the NSM framework and, according to these data, there are almost 30 world languages under scrutiny with a similar number of scholars devoted to the NSM research agenda. Finally, to partly defend legitimate criticism aimed at NSM, I want to quote Bert Peeters and Maria Giulia Marini who are engaged in exploring the potential usefulness of NSM in the field of medicine: "the ultimate aim of narrative medicine is not to replace evidence-based medicine, but to *supplement* it, with resulting benefits for both sides" (2018: 264, original emphasis) – the key conclusion being that the rigour of NSM's semantic analysis strives not to provoke misunderstandings and confusion but rather intensify and aid everyday communication. It is in this vein that the following study is understood to continue.

Wierzbicka is keen to look at NSM as a "supplementary *lingua franca* that can be used to explain ideas and meanings which go beyond scientific vocabulary and may entail such public spheres as international relations, politics, business, law, education, ethics, and can generally assist in any situation where the underlying objective is to achieve best-possible clarity in terms of what one wants to say"[1] (2017: 23). She never mentions information technology (IT) and this study aims to fill this missing area.

4. Old programming concepts in new programming languages

Out of a long list of programming concepts, a variable and a constant stand out as one of the most basic constructs that enable to organise digital data into a reusable repository. Apple reference book for Swift programming language has this to say about its modern and relatively young invention: "Swift is a new programming language for iOS, macOS, watchOS, and tvOS app development. Nonetheless, many parts of Swift will be familiar from your experience of developing in C and Objective-C" (2019: 49). Typically, as is the case with most modern programming languages, Swift builds on the core assumptions of a set of older languages, i.e. a C programming language and its slightly younger brother, Objective-C (which are still in wide use today, especially Objective-C, which was the first commercial language used by Apple). The problem with older programming languages is while they are still efficient commercially in selected areas of web services, they sometimes tend to pose a serious challenge in terms of human readability, particularly for newcomers, i.e. people with no or little technological background. In an attempt to address these issues, developers and software architects found it their duty to create programming languages whose syntax and semantics would be able to generate a familiar feeling of reading a line of code just as a sentence in a natural language (typically, English) is read. Judging how successful these noble attempts have been is not within the scope of this paper. However, it suffices to say that the IT industry is estimated to require of thousands of professionals to meet the growing needs of the market.

5. NSM explications of a variable and a constant

The case of a programming concept of variable is no exception when it comes to inheriting the used and tried programming constructs from languages originating from the 20th century. Apple plainly confirms this historical attachment: "like C, Swift uses variables to store and refer to values by an identifying name" (2019: 49). Additionally, "Swift also makes extensive use of variables whose values can't be changed. These are known as constants and are much more powerful than constants in C" (2019: 49). In other words, it is safe to assume that a variable and a constant are examples of unique labels, i.e. containers capable to storing data. It is worth noting that since the only difference between a variable and a constant is their capacity for mutability, the following discussion will focus on providing an explication of a variable first and

[1] Unless otherwise stated, this and all further translations from Polish are mine.

then, the next step would be to alter the component which points to the changing character of the two concepts.

In order for a variable (and a constant as well) to play its usual role in a software programme, it has to fulfil the following pre-requisite: "constants and variables associate a name (such as maximumNumberOfLoginAttempts or welcomeMessage) with a value of a particular type (such as the number 10 or the string "Hello")" (Apple 2019: 50). There are different naming conventions for variables and constants; however, the essentials remain unchanged – it is 'something' which contains 'something else' inside. Below is a tentative, first approximation of the NSM explication of the concept of variable:

variable:
a. there can be something inside it

The opening component (a) points to the main function of a variable (and a constant as well), i.e. in the present context, it has to, by necessity, contain some value stored inside.

Although not mentioned in the book explicitly, Apple's Swift language, just as any other programming language, allows a variable/constant to store precisely one value at a given time in the lifespan of an application/programme. Storing multiple values inside one 'label'/'container' is possible; however, it would require a completely different type of storage arrangement which Apple refers to as 'collection types'. None of this is the concern of the present discussion; however, the single-data storage capacity that a variable/constant is equipped with by default has to be included in the explication. Component (b) and component (c) are a tentative proposition to cover features specific both to a variable and a constant, i.e. it is technically impossible to have more than one value stored in a variable or a constant:

variable:
a. there can be something inside it
b. this something inside is one thing
c. this something inside cannot be two things

The formal vocabulary to use if we intend to create a variable (and a constant as well) is, according to Apple, declaring a variable and/or a constant: "constants and variables must be declared before they're used" (2019: 50). Additionally, in line with one of the underlying assumptions behind any programming language, which put heavy emphasis on conciseness and readability, also in the case of Apple's Swift language, the rule of compactness is observed unconditionally: "you declare constants with the 'let' keyword and variables with the 'var' keyword" (2019: 50). The following is a classic example which reappears in different programming tutorials:

var welcomeMessage = "Hello world"

What this crucially implies is that a developer declared a variable and named it 'welcomeMessage'. The variable contains one value (component (b)); in this case, the value is of type string: 'Hello world' (a string is a piece of text) which also offers some tangible support

for the presence of component (c) in the proposed explication. What is more, Swift language has an in-built list of specific types of values that can be passed into a variable/constant, including numbers (floats and doubles), Booleans and strings. In case of a variable which, unlike a constant, can manipulate its contents, one has to exercise extra caution not to mix different types of values in one variable: "once you've declared a constant or variable of a certain type, you can't declare it again with the same name, or change it to store values of a different type" (Apple 2019: 53). Swift is also strict in disallowing transformation of a variable into a constant and vice versa: "nor can you change a constant into a variable or a variable into a constant" (2019: 54). In light of these observations, the explication of the concept of variable requires an update:

variable (var) of kind X:
a. there can be something inside it
b. this something inside is one thing of kind X
c. this something inside cannot be two things of kind X
d. many things of kind X can be inside it at many times
e. … one thing at one time, another thing at another time

In order to account for the fact that a variable can be passed values of a specific type only (strings, floats, integers, etc.), the very title of the explication has taken a slightly different shape. Now the concept of a variable is described in a manner that implies its capacity to accept a restricted number of value types; 'variable (var) of kind X' limits the scope of values that a given instance of a variable can store. Additionally, supplementing the heading of the explication with a 'var' keyword by which Swift formally recognises that it is, indeed, a variable that is being created, provides more clarity in terms of what a programmer is working with and would be particularly helpful for those who are already familiar with the naming conventions adopted in Swift.

Alternative versions of components (b) and (c) further emphasise the point that apart from allowing precisely one value to exist inside a variable, it can accept only one value of a given type ('of kind X'). This changed design appears to correspond well with the heading of the explication and, arguably, would be the preferred version of the explication.

Since it is self-evident that the present discussion is focused on two distinct programming concepts, there appears to be no sound reason to supplement the explication with a component related to unlawful alteration of variables into constants and constants into variables. The very heading of the explication suggests that a variable is a separate and autonomous concept, which is not meant to be confused with a constant.

Eventually, components (d) and (e) point to the distinguishing feature of Swift's variables, i.e. their hardwired capacity for mutability. Values stored inside a variable can and are well expected to change in the course of an application's/programme's lifespan as long as they conform to the initial type declared at the point of declaration. It is the actions taken by the end-user that condition these changes to happen and components (d) and (e) clearly imply that a variable is open to assigning new values.

In light of the discussion above, we are now ready to try and delineate the underlying difference between a variable and a constant using NSM:

constant (let) of kind X:
a. there can be something inside it
b. this something inside is one thing of kind X
c. this something inside cannot be two things of kind X
d. at all times this one thing is always the same thing

The explication of a constant is one component shorter than the parallel explication of a variable as the essence of difference lies in the fact that, unlike in the case of a variable, a constant is principally an example of an immutable 'container' (component (d)). Constants will never change; their value (one and only one) will always remain the same ('one thing […], at many times') and since no change in value is expected, it follows that a constant keeps the same type of value ('of kind X') throughout the execution of a programme.

6. Conclusions

Academic literature discussing Natural Semantic Metalanguage provides plenty of evidence that the somewhat provocative claim of there being a set of semantic primes available in any natural language can, in fact, serve its purpose. The purpose of NSM is to make complex concepts more readily comprehensible across different languages. Concepts analysed in this paper might not fall into the category of being overly obscure or perplexing; they were chosen, first, because they are a starting point for anyone who begins their journey with programming and, second, because of their apparent simplicity, a justified assumption could be initially entertained that it was relatively easy to come up with readable and straightforward explications. This was, however, not the case, as the core explication of the concept of a variable, which shares the majority of its components with the explication of the concept of constant, underwent many changes and corrections before arriving at the final result. The question of readability proposed by the explications provided in this paper might be challenged, as unconditional reliance on 'indefinables' exclusively seems to result in explications, which could be readable and comprehensible only for readers who have already been exposed to the basic programming concepts. The vague nature of programming concepts requires a degree of abstract thinking, which can be learnt through practice. The type of reductionism or reductive paraphrase pursued here calls for a certain amount of initial programming knowledge to refer to. However, it does not exclude the usability of the proposed explications as a quick and accessible reference point for beginners.

The two explications included in this study make no claim to be exhaustive and are open to further amendments. The realm of IT seen from the viewpoint of semantics is heavily underexplored and calls for further research is more than justified. Understanding the meaning of programming concepts, such as a variable and a constant among several other more complex programming constructs, remains problematic for industry newcomers. Simple definitions, emulating a dictionary entry, definitions which could be available instantly via an app or a digital collection of NSM explications might serve as a useful aid in developing skills which are so much sought-after on the market today.

References

Apple. 2019. *The Swift Programming Language (Swift 5.1)*.

Blumczyński, P. 2013. Turning the Tide: A Critique of Natural Semantic Metalanguage from a Translation Studies Perspective. In C. O'Sullivan (ed.), *Translation Studies, Vol. 6, 3*, 261–276. London: Taylor & Francis.

Bogusławski, A. 2011. A Note on Apresjan's Concept of 'Polish School of Semantics'. With an Appendix. In A. Bogusławski (ed.), *Reflections on Wierzbicka's Explications & Related Essays*, 79–95. Warszawa: BEL Studio.

Goddard, C. 1994. Semantic Theory and Semantic Universals. In A. Wierzbicka and C. Goddard (eds.), *Semantic and Lexical Universals*, 7–29. Amsterdam/Philadelphia: John Benjamins Publishing Company.

Goddard, C. 2008. Natural Semantic Metalanguage: The State of the Art. In C. Goddard (ed.), *Cross-Linguistic Semantics*, 1–34. Amsterdam/Philadelphia: John Benjamins Publishing Company.

Goddard, C., and A. Wierzbicka. 2014. *Words and Meanings. Lexical Semantics Across Domains, Languages, and Cultures*. Oxford: Oxford University Press.

Goddard, C. 2018a. *Ten Lectures on Natural Semantic Metalanguage: Exploring Language, Thought and Culture Using Simple, Translatable Words*. Leiden: Brill.

Goddard, C. 2018b. Minimal English: The Science Behind It. In C. Goddard (eds.), *Minimal English for a Global World. Improved Communication Using Fewer Words*, 29–70. Cham: Palgrave Macmillan.

MacDonald, D. 2019. *Practical UI Patterns for Design Systems*. New York: Apress.

Peeters, B., and M. G. Marini. 2018. Narrative Medicine Across Languages and Cultures: Using Minimal English for Increased Comparability of Patients' Narratives. In C. Goddard (ed.), *Minimal English for a Global World. Improved Communication Using Fewer Words*, 259–286. Cham: Palgrave Macmillan.

Reiss, K. 1971/2000. *Translation Criticism: Potential and Limitations*, trans. by Erroll F. Rhodes. Manchester: St Jerome.

Reiss, K. 1981/2004. Type, Kind and Individuality of Text: Decision Making in Translation, trans. by Susan Kitron. In L. Venuti (ed.), *The Translation Studies Reader*, 168–179. London/New York: Routledge.

Reiss, K., and H. Vermeer. 1984/2013. *Towards a General Theory of Translational Action: Skopos Theory Explained*, trans. by Christiane Nord. Manchester: St Jerome.

Snell-Hornby, M. 1988/1995. *Translation Studies: An Integrated Approach*. Amsterdam/Philadelphia: John Benjamins Publishing Company.

Tate, B. 2010. *Seven Languages in Seven Weeks. A Pragmatic Guide to Learning Programming Languages*. Raleigh, North Carolina: The Pragmatic Bookshelf.

Wierzbicka, A. 1992. *Semantics, Culture, and Cognition. Universal Human Concepts in Culture-Specific Configurations*. Oxford: Oxford University Press.

Wierzbicka, A. 1996. *Semantics. Primes and Universals*. Oxford: Oxford University Press.

Wierzbicka, A. 1997. *Understanding Cultures Through Their Keywords. English, Russian, Polish, German, and Japanese*. Oxford: Oxford University Press.

Wierzbicka, A. 2013. Translatability and the Scripting of Other Peoples' Souls. In A. McWilliam (ed.), *The Australian Journal of Anthropology 24*, 1–21. Hoboken, New Jersey: Wiley Online Library.

Wierzbicka, A. 2014. *Imprisoned in English. The Hazards of English as a Default Language*. Oxford: Oxford University Press.

Wierzbicka, A. 2017. *W co wierzą chrześcijanie? Opowieść o Bogu i o ludziach*. Kraków: Wydawnictwo Znak.

Vermeer, H. 1989/2012. Skopos and Commission in Translational Action. In L. Venuti (ed.), *The Translation Studies Reader*, 191–202. London/New York: Routledge.

Online source

Apple. *Human Interface Guidelines*.
https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/. Access: Sept. 24, 2019.