

Unicode from a Linguistic Point of View

Yannis Haralambous & Martin Dürst


Abstract. In this paper we describe and comment, from a linguistic point of view, Unicode notions pertaining to writing. After comparing characters with graphemes, glyphs with graphs and basic shapes, character general categories with grapheme classes, and character strings with graphemic sequences, we discuss two issues: the phenomenon of ligatures that stand at the boundary between graphemics and graphetics, and the proposal for the introduction of “QID emojis” which may end up being a turning point in human communication.

1. Introduction

Unicode is a computing industry standard for the encoding, representation, and handling of text. It has been introduced in 1991 and is nowadays practically the only text encoding standard worldwide. Unicode uses architectural principles, but also has to deal with engineering reality, legacy issues, and sometimes even political considerations:

The Unicode Standard is the product of many compromises. It has to strike a balance between uniformity of treatment for similar characters and compatibility with existing practice for characters inherited from legacy encodings. (*The Unicode Standard. Version 12.0—Core Specification* 2019, p. 159)

Unicode is the first encoding in the history of computing that *instructs* the user wishing to write in various writing systems of the world: the Unicode Consortium publishes a 1,018 pages long compendium (*ibid.*)

Yannis Haralambous  0000-0003-1443-6115
IMT Atlantique & LabSTICC UMR CNRS 6285
Brest, France
yannis.haralambous@imt-atlantique.fr

Martin Dürst  0000-0001-7568-0766
Aoyama Gakuin University
College of Science and Engineering
Sagamihara, Japan
duerst@it.aoyama.ac.jp

Y. Haralambous (Ed.), *Graphemics in the 21st Century. Brest, June 13-15, 2018. Proceedings Grapholinguistics and Its Applications* (ISSN: 2534-5192), Vol. 1.
Fluxus Editions, Brest, 2019, p. 127–166. <https://doi.org/10.36824/2018-graf-hara1>
ISBN: 978-2-9570549-0-9, e-ISBN: 978-2-9570549-1-6

that is comparable, in size and in the amount of details, to the monumental *World's Writing Systems* by Daniels and Bright (1996). And furthermore, the Unicode Consortium has released 14 Standard Annexes, 7 Technical Standards, 6 Technical Reports and 4 Stabilized Reports dealing with issues as manifold as line breaking, bidirectional rendering, vertical text layout, emojis, etc.

In its almost thirty years of existence, the Unicode Consortium has patiently built a technical vocabulary to describe computing issues but also (grapho)linguistic concepts. In this paper, after an introduction to the issue of encoding text, we will consider Unicode's approach to writing, from a linguistic point of view. For this we will adopt a task-oriented approach and will compare two processes: on the one hand, a person reading and understanding text that is displayed on an electronic device, and on the other hand, a machine "reading" and analyzing text from an input flow. Common to these two processes are their extremities: both take Unicode-encoded text as input, both result in "understanding" the text, in the sense of "accessing the various linguistic levels and extracting semantics" contained in it.

The reason for comparing these two—seemingly quite different—processes is that they reveal the double nature of Unicode: in order for humans to read text on screen, Unicode has to supply sufficient information for the text to be adequately rendered (with the help of a rendering engine and information provided in fonts); in order for algorithms to "read" text from an input flow, Unicode has to supply sufficient information to convey all strata of linguistic information. These two needs are complementary and Unicode has been engineered to handle both of them.

This paper is structured as follows: in Section 2 we give a quick review of the fundamentals of grapholinguistics, an introduction to the issues underlying the encoding of text and some extreme cases of text difficult to encode. Section 3 presents the "reading" processes we will refer to in the following sections. Section 4 compares various Unicode terms with grapholinguistic notions. Section 5 deals with ligatures, and Section 6 with emojis.

2. The Context

2.1. Linguistic Fundamentals

When Martinet (1970) defined *double articulation*, he considered phonemes as the lowest level of articulation in the hierarchy of language. A *phoneme* is a distinctive unit in a system and has no meaning per se. It is

through building sequences of phonemes that elementary units of meaning emerge, which we call *morphemes*. This process is called *second articulation*. To verify whether units are distinctive, Martinet used the method of *commutation*: if by replacing a single elementary unit by another in a morpheme, its meaning changes, then these two units are indeed phonemes: “cat” vs. “hat,” “cat” vs. “cut,” “cat” vs. “car”; if not, then they are *allophones* of the same phoneme. As for morphemes, their combination gives access to higher levels of meaning through syntax, and this process is called *first articulation*.

Anis (1988, p. 89) and Günther (1988, p. 77) use the same method to define *graphemes*: in their approach, which is called *autonomistic* because it does not involve phonemics, graphemes are units of written text that have no meaning per se, but are distinctive members of a system. They are defined by commutation, and through their *catenation* (a notion introduced by Sproat (2000, p. 13), see also § 4.4) morphemes emerge: in the English language, <c>, <a>, <t> and <s> have no meaning, but their catenation as <cats> contains two morphemes: <cat>, which represents the concept of “cat,” and <s>, which represents the feature of plural number.

Distinctive units are necessary because humans live in an analog world. As an infinite variety of sounds is perceived by the ear and an infinite variety of shapes is perceived by the eye, our brain has first to select those that participate in linguistic communication (which are called *phones* and *graphs*, respectively), and then to classify them into a finite set of classes by which morphemes are formed. *Graphs* are studied by the discipline of *graphetics* (Meletis, 2015), the name of which is inspired by the analogous discipline *phonetics* which is studying sounds (called *phones*) produced by humans in the frame of oral communication.

Besides *graphs* (the shapes) and *graphemes* (the elementary linguistic units), Rezac (2009) introduces the intermediate notion of *basic shape*, i.e., clusters in the space of possible graphs representing the same grapheme, such as |a|¹ and |α| representing grapheme <a>, or |π| and |ϖ| representing grapheme <π>. Note that these basic shapes do not commute in the context of, e.g., English and Greek language (<cat> ≡ <cat> and <πρός> ≡ <ϖρός>) but can very well commute in other contexts, such as IPA notation system (where <a> and <α> represent different phones), or mathematical notation (where π and ϖ may represent different mathematical variables).

1. In this paper we use the following notation: <.> for graphemes, |.| for graphs, /./ for phonemes and [.] for phones.

2.2. Encoding Text

The Cambridge Dictionary of English defines the verb “to encode” as

to put information into a form in which it can be stored, and which can only be read using special technology or knowledge.

This definition involves three actions: “putting,” “storing,” and “reading”. In our context, “putting” will be understood as “converting analog information into digital form,” or “producing (digital) information on a digital medium,” and we will restrict ourselves to information contained in textual data, where “text” is taken in a rather broad sense, including data in various notation systems such as mathematical formulas, etc. “Storing” can be digital or analog (such as in physically printed material), and “reading” can take different forms, depending on the actor: a human can read an analog text (optically or haptically) produced by mechanical means, or read an analog text produced by a digital device (and hence using digital information), a machine can “read” an analog text (by OCR), or “read” digital data in the sense of a program receiving the data through an input stream.

We will discuss “reading” processes in the next section. In this section we will consider the form in which textual data are converted as a result of the encoding process. Text in natural language (and this is the main target of the Unicode encoding standard) is a complex object with many strata of information. Even if we restrict ourselves to information of linguistic nature, the “encoding” process can be manifold:

1. One of the most common input devices is the computer keyboard, which is functionally a descendent of the typewriter. “Encoding” a text via a typewriter amounts to *keyboarding* it. Keyboarding a text amounts to selecting keys, pushing them and obtaining a 1-dimensional graphetic sequence (Meletis, 2019, pp. 117–120) on the paper. The size of the paper being limited, the typewriter’s carriage return allows the writer to build an 2-dimensional graphetic sequence in areal space. As for the computer keyboard, it also has a carriage-return key, but its use is not necessary since computer memory can be considered as a “page of infinite width” and therefore “encoding” a text through a computer keyboard results in a long 1-dimensional sequence of elementary information units corresponding to keys (or key combinations) pushed by the keyboarder.

The result of this kind of “encoding” is a digital object called *plain text* and this is the type of data Unicode claims to encode.

2. Other legacy text production techniques such as typography have a wider spectrum of visual communication methods (italics, letterspacing, color, etc.) which can be used for various linguistic or paralinguistic functions and therefore can be considered as an integral part

of text and need to be encoded as well. Markup languages such as XHTML (Pemperton et al., 2018) or XSL-FO (Berglund, 2006) handle this kind of encoding efficiently, the result being called *rich text*.

3. Natural language has two main modalities: the written modality and the spoken modality. In languages with shallow orthography such as Italian or fully voweled Standard Arabic, one can easily convert data between these modalities, with little or no information loss; in languages with deep orthography, such as English or Greek, this process requires elaborate algorithms and heavy linguistic resources. By *phonetically annotating* an (encoded) text, one has immediate access to both modalities. A text encoded, e.g., in FoLiA format (Gompel and Reynaert, 2013) can have phonetic and/or phonological annotations.
4. Being a linguistic object, text can be analyzed using traditional linguistic methods, and the results of this analysis can be marked in the text, resulting into what is called *annotated text*. This may seem unnecessary for a human reader who is knowledgeable of the natural language of the text, but can be useful for a human reader learning the language, or for the machine having to process linguistic data. Indeed, the first step of most Natural Language Processing algorithms is a morphosyntactic parse, and obtaining the representation of a text in, e.g., CoNLL-U format (Marneffe et al., 2013) is another kind of “text encoding,” where part-of-speech tags, lemmas and dependency relations are explicitly included.
5. But why stop at the syntactic level? The next step is to perform *semantic annotation* and to encode concepts present in the text and relations between them by aligning them with ontologies, knowledge bases and other semantic resources. This is possible through Semantic Web technologies such as OWL (Bao et al., 2012) and RDF (Hayes and Patel-Schneider, 2014) embedded into the generic markup language XML (Bray et al., 2008). Encoding text in this way allows optimal processing by Natural Language Processing algorithms.

As we see, text “encoding” can be more or less elaborated and rich in information, depending on the target “reader”. When the “reader” is a human, then approaches 1 and 2 are clearly distinct from approaches 3–5. Indeed, the former provide a visual result that can be read by the human, while the latter enrich the text by adding additional information to it—even though one can invent new methods of displaying the additional information, such as interlinear annotations, special GUIs, etc. When the “reader” is the machine, there is no visual stage and the distinction becomes void.

Many large corpora adopt more than one encoding approach. For example, the *Digital Corpus of Sanskrit* (Hellwig, 2010–2019) is a digital object which can be “read” by a human in the traditional way, but also contains full morphological and lexical data. These data can be presented to the human user through a dedicated GUI or can be directly “read” by NLP

algorithms for processing of the text. The *Quranic Arabic Corpus* (Dukes, Atwell, and Habash, 2013) goes even farther and contains dependency syntax and semantic annotation information.

Sometimes the boundaries between the technologies we mentioned become blurry. For example, to state only two examples involving Unicode:

1. In Japanese and Chinese, *ruby* can add the phonetic realization of a morpheme (written in kanji/hanzi characters) using smaller characters from a syllabary (kana in Japanese, bopomofo in Chinese), the latter placed above the former, as in 会社 (“company,” pronounced かいし や *kaisba*). Even though Unicode proclaims that it encodes only plain text, it provides nevertheless three *interlinear annotation characters* for marking the begin of the base sequence, the separation between base and annotation and the end of the annotation sequence. XHTML also provides markup for ruby (the ruby element) and this is the method recommended by the W3C (see Sawicki et al. 2001, as well as Dürst and Freytag 2000). Ruby, as an annotation, is essentially phonological and morphological since, traditionally, ruby bases are morphemes and not individual characters—it therefore overlaps approaches 3 and 4.
2. There exist Unicode characters with empty visual representation. These characters carry information of morphological, syntactic or semantic nature, e.g., the SOFT HYPHEN character marks boundaries of graphical syllables (and is useful for obtaining correct hyphenation); the INVISIBLE SEPARATOR character marks consecutive symbols as being part of a list (a property of syntactic nature); and the INVISIBLE TIMES character marks consecutive symbols as being multiplied (a binary algebraic operation with very precise semantics). The information carried by the INVISIBLE SEPARATOR and INVISIBLE TIMES characters can also be represented by markup in a markup language such as MathML (Carlisle, Ion, and Miner, 2014): the apply and times elements.

The standard way of producing written text, as described in Meletis (2019, pp. 117–120), is to catenate graphs into *1-dimensional graphetic sequences* to fill *linear space*, until reaching the “page” boundary and then continuing on the next line in order to fill *areal space*. This approach, which is the standard approach of legacy typography, is used by Unicode and rendering engines. It inherently assumes that the geometry of 1-dimensional graphetic sequences—as long as there is no 2-dimensional higher structure such as a list or a table—carries no syntactic or semantic information.

There are cases where human creativity has transcended this model, and we will present three examples (cf. Fig. 1). It is legitimate to raise the

question whether these cases can be “encoded” by the machine without information loss. They are the following:

- (a) a page from Mallarmé’s “Un coup de dés jamais n’abolira le hasard” (“A throw of the dice will never abolish chance”) where the reading process is spatially and temporally structured by horizontal and vertical gaps, font size, font style and uppercasing. To achieve the visual result with the appropriate precision while keeping access to textual content, XHTML and XSL-FO are not sufficient and a markup language for describing two-dimensional vector and mixed vector/raster graphics, such as SVG (Bellamy-Royds et al., 2018) is necessary;
- (b) Apollinaire’s *calligram* “La colombe poignardée et le jet d’eau” (“The stabbed dove and the fountain”), where not only graphemes form a drawing, but text meaning and image are in constant interaction: For example, the soft and immaculate character of the dove’s wings is strengthened by the text fragments on their contours: “douces figures” (“soft figures”) and “lèvres fleuries” (“flourished lips”) and by six female given names followed by the question “où êtes-vous ô jeunes filles” (“where are you oh young girls?”). Also, the wound of the stabbed dove is drawn with the words “et toi” (“and you”). Here again a markup language such as SVG is necessary in order to place graphemes on curved paths while maintaining the linearity of the poem’s text, and to encode the correspondences between parts of the drawing and text segments. One can even envision an abstract hierarchical description of the drawing (involving the dove, its head, wings and queue, wings being made of feathers, etc.) where each element is linked to a text segment, so that the poem inherits the graphical structure of the drawing and so that we have an alignment between linguistic and pictorial hierarchical structures (see Fig. 2 for a small excerpt of such a structure). No less than that would be necessary to capture the subtle interaction between image and text;
- (c) and finally a Kufic calligraphy of a Quranic verse: ﴿لَا يُكَلِّفُ اللَّهُ نَفْسًا إِلَّا وُسْعَهَا﴾ (“God does not burden any soul beyond its capacity,” 2:286), written as a spiral starting from the lower right corner and going clockwise inwards so that the last word is in the middle of the drawing. Here, the act of recognizing the text inside the labyrinthian drawing symbolizes, in the frame of Muslim religion, the discovery of the word of God in the world, which is His book.

For such a calligraphy to have a dual text/image nature, a markup language such as SVG is again necessary, but also an ad hoc font with glyphs dynamically drawn out of generic “metagraphs” for the

LE NOMBRE

EXISTAT-IL
 autrement qu'hallucination éparsée d'agonie

COMMENÇAT-IL ET CESSAT-IL
 sourdant que nié et clos quand apparut
 enfin
 par quelque profusion répandue en rareté
 SE CHIFFRAT-IL

évidence de la somme pour peu qu'une
 ILLUMINAT-IL

LE HASARD

Choit
 la plume
 rythmique suspens du sinistre
 s'ensevelir
 aux écumes originelles
 naguères d'où sursauta son délire jusqu'à une cime
 flétrie
 par la neutralité identique du gouffre

425

(a)

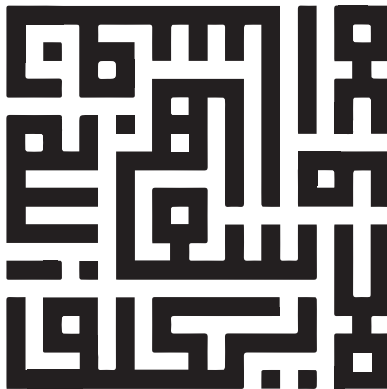
LA COLOMBE POIGNARDÉE ET LE JET D'EAU

Douces figures poi^{rdée}
 MIA Chères lèvres fleuries
 YETTE MAREYE
 ANNIE et toi LORIE
 où vous MARIE
 jeunes filles
 MAIS
 près d'un
 jet d'eau qui
 pleure et qui prie
 cette colombe s'extasie

Tous les souvenirs de nag^{uères} ? O^ù sont Reynal Billy Dalize
 O mes amis partis en guerre Dont les noms se mélancolisent
 Jaillissent vers le firmament Comme des pas dans une église
 Et vos regards en l'eau dormant Où est Crémnitz qui s'engagea
 Meurent mélancoliquement Peut-être sont-ils mort déjà
 Où sont-ils Braque et Max Jacob De souvenirs mon âme est pleine
 Derain aux yeux gris comme l'aube e^{te} jet d'eau pleure sur ma peine

CEUX QUI ONT PARTIS À LA GUERRE AU NORD SE BATTENT MAINTENANT
 Le soir tombe O sanglante mer
 Jardins où saigne abondamment le laurier rose fleur guerrière

(b)



(c)

FIGURE 1. Three examples where the size, style, position and form of graphetic sequences participate in meaning production

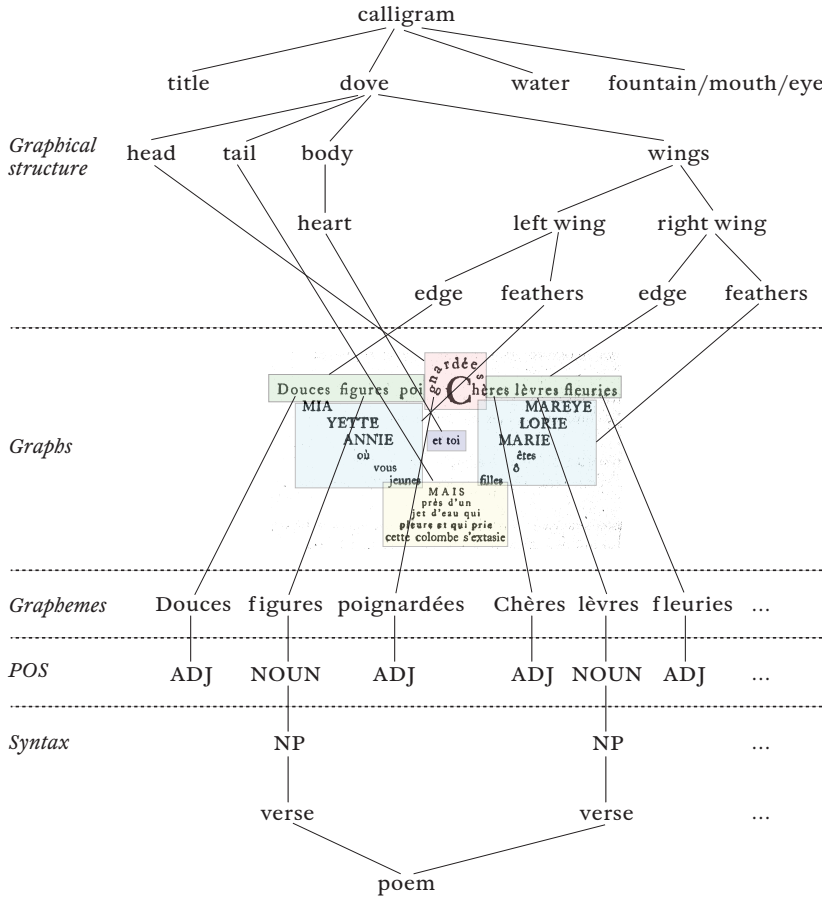


FIGURE 2. Levels of analysis of a small fragment of Apollinaire’s “La colombe poignardée et le jet d’eau”

specific textual utterance (André and Borghi, 1990; Bayar and Sami, 2010).

The aim of this section was to introduce the reader to the problematics of “text encoding,” to the various technologies available, to their mutual boundaries and overlaps, and to their limits illustrated by three examples of texts, the visual methods of which exceed the common meso/macrographetic model².

2. These terms have been introduced in Meletis (2015) and were inspired by the terms “meso-” and “macrotypography,” introduced by Stöckl (2004, p. 22).

In the remainder of this paper we will adopt a task-oriented approach and examine Unicode in the frame of three “reading” processes, differing by their actors: human or machine.

3. Three “Reading” Processes

The field of *perceptual graphetics* (Meletis, 2015) deals with the influence of the materiality of writing on perception, recognition and reading, and there has been abundant research on the particular case of perceptual graphetics of electronic devices (computer screen, tablets, smartphones, etc.). In this research domain, the displayed text is considered as a starting point and the objects of study are mainly the human perception of the signal emitted by the machine and the cognitive processes involved in recognition and understanding of textual data.

We will extend the “reading” process to the situation where the emitter and receiver are machines, and the channel is purely digital (so that no optical intermediation is involved). We therefore describe three situations where text is “read,” corresponding to three different processes, illustrated by Fig. 3 where the text consists of the morpheme “cat”:

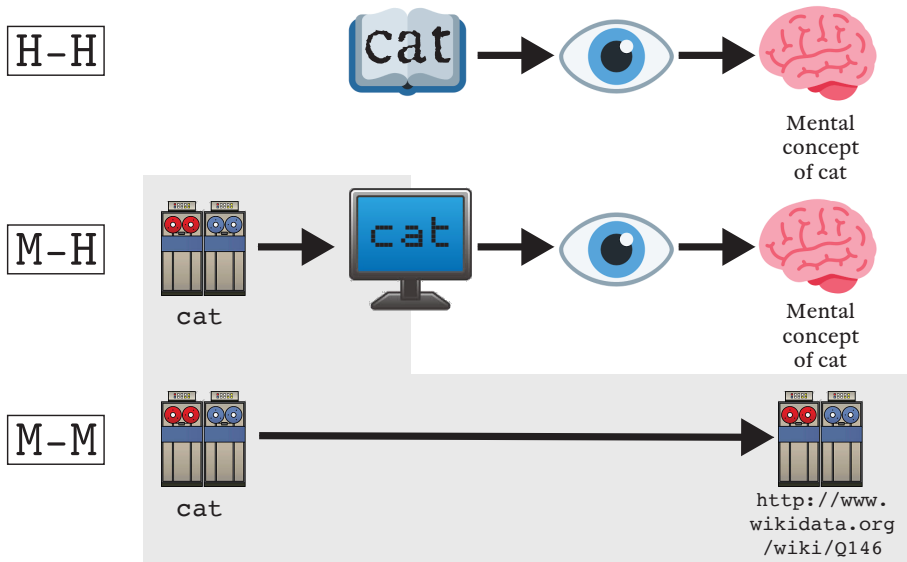


FIGURE 3. Processes $\boxed{\text{H-H}}$, $\boxed{\text{M-H}}$ and $\boxed{\text{M-M}}$ involving the morpheme <cat> and the concept of cat. Grey background denotes the digital world.

- H-H (“Human → Human”) is the process of reading from paper (either manuscript or printed): The eye sees graphs in the material form of ink on a the paper surface, the brain recognizes graphemes, combines them into morphemes and accesses the mental concepts they represent;
- M-H (“Machine → Human”) is the same process, but this time the reading surface is a computer monitor or other electronic device. The computer has the word *cat* stored, encoded in Unicode, and transmits this information to a rendering engine which extracts data from a font, builds an image, and transmits this image to a display device. At the boundary between the digital (grey background in the figure) and the analog world, the device receives the data and displays the word on its surface. The rest of M-H is the same as H-H;
- M-M (“Machine → Machine”) is the process of accessing the same concept through Natural Language Processing algorithms. The input is the same: the word *cat* encoded in Unicode. NLP algorithms have to (a) detect the current language; (b) use this information to detect words/morphemes; (c) use the context and linguistic resources to find POS tags; (d) disambiguate: is “cat” the felid? or a jazz enthusiast? or the pointed piece of wood that is struck in the game of tipcat? or is it a sturdy merchant sailing vessel?³ Once disambiguated, the intended (or most probable) concept is represented by an IRI (*Internationalized Resource Identifier*, see Dürst and Suignard (2005)) pointing to an item of the knowledge base Wikidata, namely <http://www.wikidata.org/wiki/Q146>. This item represents the concept of “cat” as a felid.⁴

The diagram of Fig. 3 obviously oversimplifies the real processes—its purpose is to show the *dual purpose* of Unicode, which has to provide sufficient information to a rendering engine to be able to correctly render graphemes on the display so that process M-H can succeed, but also to provide sufficient information to NLP algorithms for the process M-M to succeed as well.

In process M-M, Natural Language Processing algorithms need to have sufficient information to access all facets of the input considered as a linguistic object. Unicode has been engineered to allow this kind of process. But it also allows process M-H: in this case, the system transmits the string of Unicode characters to a rendering engine, which will load a *font* that maps characters to images (called *glyphs*) displayed to the reader (cf. Haralambous 2007).

3. The various meanings of the word “cat” listed here are taken from the English Wiktionary entry <https://en.wiktionary.org/wiki/cat>.

4. We have chosen Wikidata as an example, but other knowledge bases also exist, such as WordNet or Yago. See §6 for additional information on Wikidata.

Usually the choice of the font involved in the rendering process depends on the knowledge domain (computer science and mathematics publications are often typeset in Computer Modern, most other scientific disciplines in Times) or on the publisher’s graphical signature, and therefore its contribution to meaning production is secondary and mostly connotative. For that reason Unicode does not encode fonts and this information has to be added using higher protocols such as markup languages⁵ or stylesheets. Once again there are cases where human creativity has transcended this convention and has raised the choice of font to the status of important factor in the production of meaning. The reader can see an example in Fig. 4, an Austrian publicity: “Gehen Sie wählen! Andere tun es auch.” (“Go vote! Others do it too.”), where the change to a broken script in the second sentence narrows the referent of the noun “others” to right-wing extremists⁶ and thereby denotes political orientation.

For the time being, cases involving font choice or specific geometric arrangements of text can successfully go through the $\boxed{\text{M-H}}$ process but, to the authors’ knowledge, no NLP algorithm is yet capable of extracting the meaning produced by geometric and graph(et)ic features, mainly because NLP relies on the “plain text” model and discards any information of graphetic nature.

This leads us to the two main questions of this text:

1. How does Unicode model writing, in order to handle both the $\boxed{\text{M-H}}$ process and the $\boxed{\text{M-M}}$ process?
2. What are the fundamental notions of Unicode, and how do they relate to linguistic notions and processes?

In the following sections we will explore the Unicode notions of character, character category, glyph, and character string, and relate them



FIGURE 4. Austrian publicity: “Go vote! Others do it too.” (Also in Dürscheid 2016, p. 232 and Schopp 2008.)

5. The SVG (Bellamy-Royds et al., 2018) markup-language not only allows choice of font by name, but also provides XML markup for designing entire fonts which can be stored internally in the document or be used remotely.

6. Which is actually ironic, since it was Hitler who prohibited the use of broken scripts in 1943, cf. Haralambous (1991).

to the linguistic notions of grapheme, grapheme class, graph and basic shape, 1-dimensional graphetic sequence, etc.

4. Unicode

4.1. Characters vs. Graphemes

The atomic unit of Unicode is the *character*. This term has its roots in the encodings of the stone age of computing (FIELDATA in 1960, ASCII in 1963, see Mackenzie 1980 and Haralambous 2007). In the early sixties, a “character” was a “specific bit pattern⁷ and an assigned meaning”. The “meaning” was either a “control meaning” (ringing a bell, delete the previous character, etc.) or a “graphic meaning”. A “graphic meaning” was either an “alphabetic,” or a “numeric,” or a “special” (i.e., punctuation and logographs such as %, #, @, etc.). “Alphabets” were defined as “letters in the alphabet of a country” (Mackenzie, 1980, p. 16). This naive and Eurocentric approach is due to the limited use of text in computers of that period.

Unicode being a descendent of ASCII, it inherited the “character” term and introduced a panacea of additional technical terms, some of which we will try to consider in the following, from a linguistic point of view.

Probably to avoid conflict with the ancestral ASCII standard, the term “character” per se is never defined in the Unicode specification. Defined are four specializations of this term: “abstract character,” “encoded character,” “deprecated character,” and “noncharacter”.

According to the Unicode specification, an *abstract character* is defined as

a unit of information used for the organization, control, or representation of textual data. (D7 in U§3.4⁸)

We can’t help notice that this definition takes the notion of “textual data” and its perimeter for granted. This comes as no surprise since the notion of “text” is polysemic and depends on the disciplinary context. In the *Cambridge Dictionary of Linguistics* (Brown and Miller, 2013), “text” is defined as follows:

7. On the lowest level of computer memory, *bits* are binary values b_i , their concatenation $b_n b_{n-1} \dots b_1 b_0$ allows the representation of integer numbers through the formula $\sum_{i=0}^n 2^{b_i}$. A “bit pattern” is a sequence of bit values, for example 01100001 corresponds to number 97.

8. In the following we will denote, for the sake of brevity, a section ** from the Unicode Standard Version 12.0 (*The Unicode Standard. Version 12.0—Core Specification* 2019) by “U§**”.

The term originally denoted any coherent sequence of written sentences with a structure, typically marked by various cohesive devices. It has been extended to cover coherent stretches of speech.

But then again, in the same dictionary, there are no entries for “written sentence” and for “writing,” used in this definition. A “sentence” is defined as

the largest unit handled by grammar,

and “grammar” is defined as either (in the narrow sense) the “morphology and syntax of a language,” or (in the broad sense) the “morphology, syntax, phonology, semantics and even the pragmatics of a language”.

A more general definition of “text” is given in Wikipedia:

In literary theory, a *text* is any object that can be “read,” whether this object is a work of literature, a street sign, an arrangement of buildings on a city block, or styles of clothing. It is a coherent set of signs that transmits some kind of informative message,

followed by a reference to Lotman (1977). If we apply this definition to processes $\boxed{\text{M-H}}$ and $\boxed{\text{M-M}}$ we come to the fact that the purpose of text is to be “read,” be it by a human or by a machine. “Reading,” in our case, comes down to:

1. detecting and identifying the text’s elementary units,
2. applying second articulation to extract morphemes from their combination, and then
3. applying first articulation to obtain meaning from the combination of morphemes.

Our statement is that these three operations apply not only to process $\boxed{\text{M-H}}$, but also to process $\boxed{\text{M-M}}$. To start, the machine is informed by various mechanisms that it is “reading” Unicode characters and not, for example, pixel data. It is also informed on the way of reading these data, in order to convert them appropriately to elementary text units⁹. Once the machine is aware of the fact it is reading Unicode characters, identifying them becomes possible through the notion of “encoded character”:

An *encoded character* is an association (or mapping) between an abstract character and a code point, (U§3.4)

where a *code point* is defined as follows:

9. The details on the various ways of storing elementary text units on the machine level, that is using bits and bytes, have no incidence on the linguistic study of Unicode. The interested reader is invited to read U§2.5 and U§2.6.

A *code point* is any value in the Unicode codespace (U§3.4)

and the *Unicode codespace* is

the range of integers $\{0, \dots, 1,114,111\}$. (U§3.4)

In other words, in the frame of process $\boxed{\text{M-M}}$, step 1 of the “reading” process, namely *identification of elementary units* (called “encoded characters”) is *trivial* for the machine, since they are represented in memory by unique numbers¹⁰. No extra effort is required.

Not so for the human in the frame of process $\boxed{\text{M-H}}$, where elementary units are distinctive elements of a system and, as such, must be recognized by the readers, provided they are knowledgeable of the corresponding writing system or notation system. Here, the “elementary unit” (of text) corresponds to the linguistic notion of *grapheme*, as defined by Anis (1988) and Günther (1988).

So how do the notions of “character” (abstract or encoded) and “grapheme” compare?

The Unicode Consortium avoids taking position in favor or against the autonomistic approach and avoids using the term “grapheme”. Indeed, in the Unicode specification it is stated that

A character does not necessarily correspond to what a user thinks of as a “character” and should not be confused with a *grapheme*. (U§3.4)

As a critique to this statement we argue that:

- in the frame of process $\boxed{\text{M-H}}$, a character is *exactly* the information that, after being channeled through rendering engines, allows the human to recognize a grapheme without ambiguity, so it functionally corresponds to a grapheme;
- in the frame of process $\boxed{\text{M-M}}$, a character is exactly the information that is necessary to the machine to perform natural language processing, similarly to graphemes that are the information necessary to the human to process language,

and therefore one can conclude that, functionally, the notions of character and grapheme are quite close.

Nevertheless characters do not always represent graphemes. The main discrepancy between the two notions is due to the fact that some scripts (such as the Latin script or the Cyrillic script) are used for more

10. In some sense, encoded characters can be considered as signs, code points being their signifiers and abstract characters their signifieds. As with Saussurean signs, the relationship between signifier and signified is arbitrary, in the sense that there is no reason why the abstract character LATIN LETTER A is represented by code point 97.

than one language, and Unicode targets *all* languages simultaneously. To take two examples, according to Grzybek and Rusko (2009, p. 33), there is a <ch> grapheme in the Slovak language, and according to Wmffre (2008, p. 598), there is a <c'h> grapheme in the Breton language—none of these is a Unicode character. This comes from the fact that Unicode is bound to follow conventions such as national encodings or keyboard layouts, and there has never been a Slovak encoding or keyboard featuring <ch> or a Breton encoding or keyboard featuring <c'h>.

There are even Unicode characters that do not correspond to graphemes in *any* language, such as invisible Unicode characters (various spaces, SOFT HYPHEN, etc.), pictographs, emojis, etc. Also there is a large amount of characters that are graphemes in one language but not in another language using the same script (such as <č> used in Czech but not in German). More interesting is the case of characters which are graphemes in language A, are not graphemes in language B, but can still be used in B as allographs of some grapheme. For example, the letters <ڪ> and <ك> which commute in Sindhi: <ڪنڊ> (“ear”) ≠ <ڪنڊ> (“sugar”); in Arabic and other Arabic-script languages they are just allographs of the Arabic <ك>, and hence do not commute: <ڪتاب> ≡ <ڪتاب> (both “book” in Arabic).



FIGURE 5. French logos using allographs that are graphemes in other languages

This phenomenon is easier to observe in graphic design, where the use of allographs is a creative design method. In the French-language examples of Fig. 5 one can observe the use of allographs <İ>, <Ē>, <Ā>, and <í>, instead of <I>, <É>, <Ā> and <i>; these allographs are graphemes in other languages (Turkish for <İ>, Latvian for <Ē> and <Ā>, Italian, Spanish and elsewhere for <í>) and therefore are accessible to the designer through Unicode-encoded fonts.

4.2. Glyphs vs. Graphs and Basic Shapes

Meletis (2015, p. 117) defines *graphs* for the German language as follows:

Die kleinste, nicht weiter durch Leerstellen getrennte Einheit ist der Graph (dies gilt insbesondere für Druckschrift und nur eingeschränkt für

Handschrift), der jeweils einen einzigen segmentalen Raum ausfüllt und im alphabetischen Schrifttyp durch Buchstaben, aber auch nicht-alphabetische Graphen wie Interpunktions- und Sonderzeichen sowie Ziffern verkörpert wird.¹¹

This definition can easily be applied to scripts with separated atomic units (alphabetic scripts, South-East Asian scripts, Chinese script). In the case of scripts systematically connecting atomic units (Arabic and Syriac scripts, Devanagari, etc.) the condition of “smallest entity not separated by blank spaces” cannot be applied. In the case of printed text, one can refer to the historical segmentation of the connected script into types to obtain a (not perfect but reasonable) solution to the problem how to segment a shape into its constituent graphs. In the case of handwriting there is no clear segmentation and fuzzy logic has to be applied to the mapping of each part of the drawing to individual graphs contained in it.

The advantage of this definition of *graph* is that it doesn't presuppose knowledge of graphemes and of higher linguistic units: one can take a printed text in an unknown language, subdivide it into elementary graphical units and move on to the next graphetic levels (1-dimensional graphetic sequences in linear space, 2-dimensional graphetic sequences in areal space, page).

In the Unicode Standard there is no proper definition of *glyph*. The closest we can get to obtain a definition would be through the following sentence:

Glyphs represent the shapes that characters can have when they are rendered or displayed. (U§2.2)

What is understood in this definition is that the glyphs must be rendered or displayed in such a way that the characters they represent can be visually recognized by readers knowledgeable of at least one language in which the characters are used.¹²

All occurrences of the term “glyph” in the Unicode Standard refer to shapes obtained by rendering characters. This excludes shapes drawn by hand in a drawing application, and any text obtained by a means different than Unicode characters rendered by a rendering engine.

11. “The graph is the smallest entity [of the model] that is not separated by blank spaces (this is valid predominantly for printed text and only partially for handwritten text) and that occupies a single segmental space. In alphabetic writing systems, graphs are materialized by letters but also by non-alphabetic signs such as punctuation and special signs and digits.”

12. Technically the rendering of an arbitrary Unicode character is provided by rendering engines, which use data from fonts, and in font technologies there is absolutely no restriction on the shape that can be used for a specific Unicode character: one can easily create a font rendering the character LATIN CAPITAL LETTER A by the glyph |B|.

So, if we stick to this excerpt of the Unicode Standard, then (a) glyphs are an aspect of characters for which Unicode takes absolutely no responsibility, (b) absolute freedom is granted to font vendors to render characters as they like and (c) only the degree of commercial success of a font can determine the legitimacy of its glyphs as representatives of given characters. This may seem an overstatement for the common scripts, but becomes a real problem for rare scripts for which only very few fonts exist: Dürscheid (2018, §4) qualifies the Unicode Consortium as a “gatekeeper” of characters, in a similar way the font industry becomes the “gatekeeper” of glyphs of rare scripts¹³.

Fortunately Unicode avoids this anarchy situation by introducing an additional notion: the one of *representative glyph*:

The identity of a character is established by its character name and *representative glyph* in the code charts.

A character may have a broader range of use than the most literal interpretation of its name might indicate; the coded representation, name, and representative glyph need to be assessed in context when establishing the identity of a character. For example, FULL STOP can represent a sentence period, an abbreviation period, a decimal number separator in English, a thousands number separator in German, and so on. The character name itself is unique, but may be misleading.

Consistency with the representative glyph does not require that the images be identical or even graphically similar; rather, it means that both images are *generally recognized to be representations of the same character*. Representing the character LATIN SMALL LETTER A by the glyph “X” would violate its character identity. (U§3.3, emphasis introduced by us)

Representative glyphs for all Unicode characters can be found in the Unicode Code Charts¹⁴. The notion of representative glyph is very interesting because

1. it reveals the insufficiency of the intensional description of characters;
2. it induces an operational definition of glyphs: *a glyph is a shape that is generally recognized to be a representation of a character*. This definition still involves Unicode characters, but not the rendering process anymore;
3. it shows that the relation between characters and glyphs has a socio-linguistic facet: a glyph represents a given character if and only if there is a *community of people* recognizing it as such.

The notions of *graph* in linguistics and *glyph* in Unicode may intuitively seem equivalent, but the ways they are defined makes them difficult to

13. In the sense that a Unicode user without the necessary competency for creating a font with the appropriate glyphs is forced to use glyphs provided in existing fonts.

14. <https://www.unicode.org/charts/>.

compare: *graphs* are defined as units in a graphetic system, while *glyphs* are defined as socially recognizable renderings of a given character.

It is interesting to note that in Unicode, kanji/hanzi characters have not one but as many as six representative glyphs, corresponding to graphs used in China, Hong Kong, Taiwan, Japan, Korea and Vietnam. For example, the character 伶 with codepoint 4F36₁₆ (“clever,” “actor”) is presented in the following way in the Unicode code chart:

As the reader can see there are three shape families: (a) the first and sixth graphs (China and Vietnam) have a drop-like *diǎn* stroke |丿| under the “roof”; (b) the second and third graphs (Hong Kong and Taiwan) have a straight horizontal *béng* stroke |一| under the “roof”; (c) the fourth and fifth graphs (Japan and Korea) have a different lower-right component, consisting of a *béng-zhè-gōu* stroke |𠃉| and a *shù* stroke |丨| (see Haralambous 2007, pp. 154–155 and Myers 2019, pp. 13–14).

We claim that these three graphs belong in fact to three different *basic shapes*, in the sense of Rezec (2009). As they are obtained by the use of different fundamental strokes, the graphs will necessarily remain different in all possible realizations belonging to three disjoint clusters. There will never be “intermediate” cases since the fundamental strokes have to be recognizable by design as distinctive parts of the system.

4.3. Character General Categories vs. Grapheme Classes

Let us now turn to issues of classification. In the usual classification of graphemes into *logograms* and *phonograms*, the latter are defined by their relation to speech. Dürscheid (2016, p. 74) defines phonograms as follows:

Phonogramme (= Lautzeichen) sind Zeichen, die ausschließlich auf die lautliche Ebene des Sprachsystems bezogen sind.¹⁵

Such a definition is not compatible with Anis’s autonomistic approach, which considers writing without any a priori relation to speech. Anis (1988) divides graphemes into three classes, namely *alphagrams*, *topograms* and *logograms*. His definition of an *alphagram* is as follows:

ces unités distinctives, dénuées de sens par elles-mêmes, sont les composantes des unités significatives. Comme les phonèmes, les alphagrammes relèvent de la *seconde articulation*.¹⁶

15. “*Phonograms* (= Signs for sound) are signs that refer exclusively to the oral level of the language system.”

16. “These distinctive units, meaningless per se, are components of significative units. Like phonemes, alphagrams are part of *second articulation*.”

A *topogram* (Anis, 1988, p. 116) is essentially punctuation: topograms contribute to the structure and segmentation of sequences of alphagrams and logograms. As for *logograms*, they are global units having a signified (ibid., p. 139)¹⁷.

Typical examples of alphagrams are members of alphabets, of abjads, of abugidas, of syllabaries. Typical examples of logograms are graphemes such as <&>, <§>, currency signs <\$>, <€>, etc., mathematical symbols <5>, <∇>, etc., general symbols <☉>, <σ>, <ℵ>, etc.

Unicode provides a similar classification of characters in the form of a mandatory normative¹⁸ property of characters, called *general category* (U§4.5). This classification is quite different from the linguistic one:

1. all alphagrams belong to general category “L” (for “Letter”), with the following subcases: “Lu” (“uppercase”), “Ll” (“lowercase”), “Lt” (“titlecase”), “Lm” (“modifier”) or “Lo” (“other”). The general category “L” is the most populated in Unicode: it amounts to 89.84% of the total set of characters. Among them, 96.13% are caseless and therefore belong to category “Lo” (caseless alphabets, abjads, abugidas and syllabaries, and most importantly, all Chinese characters);
2. many logograms¹⁹ such as <&>, <@>, <%>, etc., are of Unicode general category “Po” (“punctuation, other”), a contradiction to their linguistic classification;
3. in the case of mathematical symbols, Unicode uses two general categories: “N*” for numbers, and “Sm” for other mathematical symbols, such as <+>, <≤>, etc. The “N*” general category contains the subcategories “Nd” (decimal digits), “No” (fractions, numbers larger than 9, circled or parenthesized numbers) and “Ni” (Roman, Hangzhou, Bamum, Greek acrophonic, Gothic, Old Persian and Cuneiform numerals);
4. The general category “So” (“symbol, other”) is a catch-all. It includes symbols such as <©>, <°>, <⊕>, but also emojis, musical symbols, technical drawing symbols, circled or parenthesized letters/ideograms/syllables, box drawing symbols, Braille patterns, Chinese radicals, Chinese fundamental strokes, hexagrams, Phaistos disk signs, sign-writing gestures, Mahjong and domino tiles, playing cards, alchemical symbols, as well as the single (!) character ARABIC LIGATURE BISMILLAH AR-RAHMAN ARRAHEEM

17. Anis does not distinguish between the iconic and the indexical semiotic function and therefore considers pictograms as being a special case of logograms. We will not adopt his choice and will consider pictograms as being distinct from logograms, even though the distinction can sometimes be blurry.

18. In the sense that Unicode-compliant software has to respect it.

19. Other than Chinese characters, which are not pure logograms since they can have different amounts of semanticity and phoneticity, cf. Haralambous (2013). As already mentioned, Chinese characters belong to category “L” (“letters”).



representing the Arabic sentence “In the name of Allah, the Beneficent, the Merciful”. Category “So” characters represent 5% of the total number of characters.

The linguistic classification of graphemes and the Unicode classification of characters differ in their finalities:

- the former focuses on the way graphemes contribute to meaning extraction: alphagrams are part of second articulation, and hence meaning emerges from their catenation; topograms structure grapheme sequences, and hence serve on the syntactic level; logograms represent morphemes;
- the latter focuses on the way characters are used by software: characters that serve in linguistic processes (“letter” category) are separated from punctuation, from mathematical symbols, and from symbols in general (among which numerous emojis). General categories are used in texts such as the *Unicode Standard Annex on Text Segmentation* (Davis, 2019b), which defines the boundaries of a “word” or of a “sentence” using general categories, or the *Unicode Standard Annex on Line Breaking* (Heninger, 2019), which gives guidelines to line breaking algorithms, based on general categories.

4.4. Character Strings vs. Grapheme Sequences

A text rarely consists of a single grapheme²⁰. Most often humans produce *sequences* of graphemes. Contrary to phonemic input which is linear due to the structure of human speech production organs, grapheme sequences are usually materialized on a 2-dimensional surface. The linear order of phonemes is often represented by a similarly linear order of graphemes (like the ones the reader is reading at this moment), but there are exceptions. A nice example is the Khmer script: the sequence of graphemes representing phonemes /kk/ is written <ក្ក> and when one adds an additional grapheme representing the /r/ phoneme, the sequence representing the phonemic sequence /kkr/ is written as <ក្ករ> (the <រ> grapheme is written to the left of the previous ones), and if one adds a vowel /iə/ this will surround the preceding graphemes: <ក្ករី> (example taken from Haralambous 1994b).

In linguistics, grapheme sequences have been studied by Sproat (2000), in the frame of generative phonology theory introduced by

20. As always, there are exceptions to this rule, such as the title of the recently published book 心 (Kazuo, 2019).

Chomsky and Halle (1968). In this theory one admits the existence of two levels of representation of phonological data: the *underlying form* and the *surface form* (with the possibility of any number of intermediate levels). The latter is obtained by sequentially applying phonological rules to the data of the former (every intermediate level being the output of some rule). A sequence of rule applications going from underlying to surface level is called a *derivation*. Sproat (2000) states that *graphemes can be obtained by using derivations from the same underlying representation as phonemes*, i.e., the graphemic surface representation can be obtained by derivations of the same underlying representation used to obtain surface phonemes. Sproat furthermore claims that this derivation is a *regular relation* in the sense of finite state transducers (Kaplan and Kay, 1994), and that it is consistent throughout the vocabulary of a given language.

Regular relations are context-free, therefore if γ is a derivation and $a \cdot b$ is the catenation of two underlying representation units, then $\gamma(a \cdot b) = \gamma(a) \cdot \gamma(b)$. In fact, according to Sproat (2000), we have not a single but five *catenation operators*, namely $\vec{\cdot}$, $\overleftarrow{\cdot}$, \downarrow , \uparrow , and \odot , representing placement of the second grapheme on the right, on the left, underneath, on top of, or around the first grapheme.

For example, the derivation rules for Korean hangul are as follows (ibid., p. 43):

1. for syllables σ_1 and σ_2 , $\gamma(\sigma_1 \cdot \sigma_2) := \gamma(\sigma_1) \vec{\gamma}(\sigma_2)$;
2. for onset-nucleus ωv and coda κ , $\gamma(\omega v \cdot \kappa) := \gamma(\omega v) \downarrow \gamma(\kappa)$;
3. when the coda κ is complex: $\kappa = \kappa_1 \cdot \kappa_2$, then $\gamma(\kappa_1 \cdot \kappa_2) := \gamma(\kappa_1) \vec{\gamma}(\kappa_2)$;
4. for onset ω and nucleus v , either
 - (a) $\gamma(\omega \cdot v) := \gamma(\omega) \vec{\gamma}(v)$, when v belongs to the vertical jamo class, or
 - (b) $\gamma(\omega \cdot v) := \gamma(\omega) \downarrow \gamma(v)$, when v belongs to the horizontal jamo class;
5. (rule added by us) sometimes the nucleus v is complex and hence can be written as $v = v_1 \cdot v_2$ where v_1 is horizontal and v_2 is vertical, then we first apply rule 4(a) to $\omega v_1 \cdot v_2$ and then rule 4(b) to $\omega \cdot v_1$.

As an illustration, let us apply these rules to Hangul syllable $\langle \text{ㅏ} \rangle$: it consists of an onset $\langle \text{ㄱ} \rangle$, a nucleus containing two jamos $\langle \text{ㅏ} \rangle$ and $\langle \text{ㅣ} \rangle$ of which the first is horizontal and the second vertical, and a coda consisting of two jamos $\langle \text{ㄷ} \rangle$ and $\langle \text{ㅎ} \rangle$. According to rule 5, we first apply rule 4(a) to $\langle \text{ㄱ} \text{ㅏ} \rangle \cdot \langle \text{ㅣ} \rangle$ to get $[[\langle \text{ㄱ} \text{ㅏ} \rangle] \vec{\langle \text{ㅣ} \rangle}]$ and then rule 4(b) to $\langle \text{ㄱ} \rangle \cdot \langle \text{ㅏ} \rangle$ inside it, to obtain $[[\langle \text{ㄱ} \rangle \downarrow \langle \text{ㅏ} \rangle] \vec{\langle \text{ㅣ} \rangle}]$. Then we apply rule 3 to the coda $\langle \text{ㄷ} \rangle \cdot \langle \text{ㅎ} \rangle$ to obtain $[\langle \text{ㄷ} \rangle \vec{\langle \text{ㅎ} \rangle}]$, and finally rule 2 to join onset-nucleus and coda, in order to obtain

$$[[\langle \text{ㄱ} \rangle \downarrow \langle \text{ㅏ} \rangle] \vec{\langle \text{ㅣ} \rangle}] \downarrow [\langle \text{ㄷ} \rangle \vec{\langle \text{ㅎ} \rangle}]$$

as decomposition of $\langle \text{ㅏ} \rangle$. Sproat calls this kind of formal grammar, a *planar regular grammar*.

Among the various applications of these rules there is also diacritization: the grapheme $\langle \hat{a} \rangle$ can be represented by $\langle \text{a} \rangle \uparrow \langle \hat{\ } \rangle$.

It is noteworthy that planar catenators can be applied on all graphic levels: inside a grapheme, between grapheme and diacritic, between graphemes to form morphemes, between morphemes to form lines of text, between lines of text to form paragraphs and pages, between pages to form books, similarly to the graphetic model of Meletis (2015).

In Unicode, there are two notions corresponding to the linguistic notion of grapheme sequence:

1. *combining character sequences*, where we deal with a single “base” character and one or more diacritics, and
2. *character strings*, where we deal with more than one base character.

In the first case, we use the operation of *combination*: a character, which has to be of category other than “M” (“combining mark”) is followed by one or more *combining characters*, i.e., characters of category “M”. For example, to obtain the rendering |â| one can use two characters: LATIN LETTER A followed by COMBINING CIRCUMFLEX ACCENT. In other words: when rendering this sequence of Unicode characters, Unicode-compliant software has to place the glyph of the circumflex accent upon the glyph of the character preceding it.



Combination is a very powerful feature because one can combine any sequence of combining characters (there are 2,268 of them) with any of the 121,490 graphic characters, which results in an astronomical number of combinations. *Interscript* combination is not very frequent but it may happen, as in the logo of the popular Japanese coffee chain “Saint-Marc Café” <サンマルクカフェ> where the last kana carries an acute accent as in the “é”

of the French word “Café,” which is transcribed: the French diacritic is transplanted into the kana syllabary.

Not all combining marks are placed on the same position relatively to the base character, and there are no less than 54 classes of combining characters with respect to the relative position of the diacritic. Such classes are “Above” as in <â>, “Kana_voicing” as in <ポ>, etc.

The rendering of combining character sequences is the responsibility of rendering engines, which combine glyphs in a very precise way, using information stored in the font, namely attachment points placed around glyphs by the font designer.

The second case of character sequencing is the one of *character string*. A character string is a sequence of characters. The order that must be given to characters to obtain graphotactically correct grapheme sequences in the frame of the M-H process is called *logical order*. As U§2.2 puts it:

The order in which Unicode text is stored in the memory representation is called *logical order*. This order roughly corresponds to the order in which text is typed in via the keyboard; it also roughly corresponds to phonetic order.

As hinted by the word “roughly,” there are exceptions to this definition, the most notorious one being the encoding of Thai and Lao scripts: to represent the /ke:/ syllable in Khmer, the logical order agrees with the phonetic order and places the character KHMER LETTER KA <ក> before the character KHMER VOWEL SIGN E <ឺ>, even though the grapheme of the latter is on the left of the grapheme of the former: <ឺក>; to obtain the analogous grapheme sequence in Thai or in Lao, the logical order is to place the character THAI CHARACTER SARA E <เ> (resp. LAO VOWEL SIGN E <ື>) *before* the character THAI CHARACTER KO KAI <ค> (resp. LAO VOWEL KO <ກ>): <คเ> (resp. <ກື>) and not *after* the consonant as in Khmer. In other words, logical order agrees with phonetic order in Khmer, but *not* in Thai and Lao, even though these scripts are historically very closely related. The reason is compatibility with preexisting Thai/Lao encodings and typewriter practice.

The greatest advantage of Unicode’s “logical order” is that it solves—at least in computer memory—the problem of mixed left-to-right and right-to-left scripts, such as Latin and Arabic (or Hebrew, or Syriac). In memory, both Latin and Arabic characters are stored in phonetic order. The difficulty arises when such mixed texts have to be displayed. For that, Unicode attaches a default (horizontal) direction to every character: Latin characters have default left-to-right direction (even though Da Vinci wrote the other way around) and Arabic characters have default right-to-left direction. The *Unicode bidirectional algorithm* (Davis, 2019c) provides the order of glyphs for character strings containing characters with different default directions. Because of nested phrases and punctuation marks without default direction, the bidi algorithm sometimes fails to provide the correct result. In that case, the user can insert special characters, such as RIGHT-TO-LEFT EMBEDDING and POP DIRECTIONAL FORMATTING, which will change the algorithm’s output. Here is an example: in the sentence <Did he say “Welcome”?> the question mark is placed outside the quoted <“Welcome”> because it belongs to the noun phrase <Did he say...> and not to the quoted welcome greeting. Translating <“Welcome”> into Hebrew, one gets:

|Did he say ?“ברוך הבא”|,

where the question mark is placed to the left of the quoted phrase, while it should be placed to its right, as it applied to the whole “Did he say ***?” sentence. We avoid this by inserting a LEFT-TO-RIGHT MARK character just before the question mark, resulting in the correct rendering:

|Did he say “ברוך הבא”?|.

This problem would be avoided if there were two distinct exclamation marks in Unicode (a left-to-right one and a right-to-left one), which is

not the case. Only those punctuation marks that have different basic shapes in the two directions have their right-to-left counterparts included in Unicode, e.g., ARABIC QUESTION MARK <؟>, ARABIC COMMA <،> and ARABIC SEMICOLON <؛>.

We can conclude that the formal approach of Sproat (2000) can represent both Unicode combining sequences and character strings, but lacks fine details such as the 54 combining character classes, etc. On the other hand, Unicode logical order comes in handy for people to know in which order they have to type characters, even though it suffers from inconsistencies due to compatibility with legacy encodings. Finally, the Unicode bidirectional algorithm is a good solution for encoding character strings of scripts in different directions, but one needs to take care of ambiguities due to nesting of phrases and to neutral-direction punctuation marks.

5. Ligatures

When a character string is handed over to a rendering engine as input of the [M-H] process, in most cases Sproat's regularity principle applies, so that the derivation of a sequence of adjacent underlying linguistic units is simply the planar catenation of the derivations of individual units. There are nevertheless language-dependent exceptions to this principle, namely *ligatures*.

Ligatures are graphs obtained by merging adjacent graphs. They can be *optional* or *mandatory* (optional in the sense that the adjacent graphs may also, under some conditions, remain unchanged), and their use may or may not be taken into account in linguistic analysis.

- *Mandatory ligatures* are those occurring systematically when two given graphs are adjacent. The most prominent example is the Arabic *lam-alif* |لا| (compare with the hypothetical unligatured *|لا|). The use of the *lam-alif* ligature is a fundamental rule of the Arabic writing system from its very beginnings, as in the following sentence typeset in undotted 6th century CE *Mashq Kufi* (Mousavi Jazayeri, Michelli, and Abulhab, 2017):

لا رأيت ولا سمعت

لا رأيتُ ولا سمعتُ (“I have neither seen, nor heard”). The *lam-alif* ligature is used in *all* Arabic-script languages. It is also noteworthy that it has been taught for centuries in schools as being part of the Arabic alphabet (Dichy, in this volume) and that Arabic typewriters contain a key for it, even though it is not considered as a letter of the Arabic alphabet. Nevertheless, despite its universal presence in the Arabic

script, *lam-alif* remains a ligature and hence there is no *lam-alif* Unicode character²¹.

- *Discretionary ligatures* are those that occur under certain conditions when two given graphs are adjacent. Their use may or not have an incidence on linguistic layers.

We subdivide discretionary ligatures into two classes: *esthetic* ligatures and *linguistically motivated* ligatures (cf. Haralambous 1995):

- *Esthetic ligatures* only contribute to legibility and esthetic quality of the written text. Typical examples are the Latin |fi|, the Arabic |فح| and the Armenian |փւ| (compared with the unligatured |fi|, |مف| and |փւ|). The reasons for using esthetic ligatures are purely visual: to avoid overlapping of bulb and dot in the case of |fi|, to compress text by writing |فح| vertically, to avoid excessive blank space between graphs in the case of |փւ|.

It should be noted that even though esthetic ligatures have no linguistic motivation, their use may be language-dependent. For example, Turkish language does not use ligatures |fi| and |ffi| because the Turkish graphemic system has graphemes <i> and (dotless) <ı>, and the use of the ligatures would cancel their distinctive potential and introduce ambiguity.

- *Linguistically motivated ligatures* have an ambiguous status between stand-alone graphemes and grapheme sequences. Typical examples are the French <œ> and the Dutch <ij>. Both have a grapheme-like behavior when it comes to case, since they are uppercased as stand-alone graphemes: <Œttingen>, <Umegen> (and not *|Oettingen| or *|Ijmegen|). On the other hand, and unlike the Arabic *lam-alif*, they do not appear on typewriters²². They can be qualified as second-class citizens of the graphemic system: they do not appear in prescriptive grammars, are hardly taught in school, and are difficult to access on computer keyboards.

The distinction between esthetic and linguistically involved ligatures can be blurry: for example, in the German language, the f-ligatures are *indirect morphological markers* since they are only used intramorphemically. In German typographic practice, ligatures crossing morpheme boundaries are avoided: |Kaufleute|, |Auffassung|, etc.

21. In fact there is a “presentation form” character ARABIC LIGATURE LAM WITH ALIF ISOLATED FORM, but its usage is highly discouraged: “[Presentation form characters] are included here for compatibility with preexisting standards and legacy implementations that use these forms as characters. Instead of these, letters from the [standard] Arabic block should be used for interchange”. (UŞ9.2)

22. But they were present on the keyboards of localized Monotype/Linotype typesetters in the late 19th and most of the 20th century.

Ligatures are interesting from a theoretical point of view because they challenge the definition of script as a system of distinctive elementary units that allow double articulation. As Nehrlich (2012, p. 30) writes:

Die Ligatur stellt die Hauptmerkmale des Schriftsystems in Frage: Die wohl grundlegendste Anfechtung besteht in der Tatsache, dass das Vorhandensein von Ligaturen das Konzept des Buchstabens problematisiert. Buchstaben sind die Grapheme, aus denen das Alphabet besteht, doch taugen sie ausschließlich als abstrakte Vorstellung. Sobald es um die materielle Realisierung von Schrift geht, verliert der Begriff des Buchstabens an Gültigkeit: Das Vorkommen von Ligaturen falsifiziert die Bestimmung von Buchstaben als das, was innerhalb eines Wortes durch Lücken getrennt ist.²³

Indeed, from a systemic point of view, (esthetic) ligatures are unnecessary since they do not carry any linguistic information, and unnecessary features tend to disappear in an evolving system. But ligatures happen to exist for as long as writing exists and do not seem to face a risk of extinction in the near future. Ligatures make us realize that, just like light has a dual particle/wave nature, graphemes also have a dual nature since they carry both graphical and linguistic information. Similarly to Young's double-slit experiment that has revealed the dual nature of light, ligatures reveal (at least in Western languages) the dual nature of graphemes.

The dual nature of graphemes (and hence also of Unicode characters that represent them in the digital world) is the core difference between M-H and M-M processes, and it comes as no surprise that the Unicode Consortium has very carefully examined the issue of ligatures.

Indeed, Unicode draws a clear line between linguistically motivated ligatures on the one side, and esthetic or mandatory ligatures on the other. The former are first-class citizens of the encoding (for example, <œ> is encoded as LATIN SMALL LIGATURE OE and <ij> as LATIN SMALL LIGATURE IJ). This is not the case of for esthetic and mandatory ligatures²⁴.

Esthetic ligatures are handled by rendering engines, but the user can prevent their use by introducing a special "ligature-breaking" character, called ZERO WIDTH NON-JOINER. This is, for example, what is needed in German language to avoid intermorphemic ligatures.

23. "Ligatures challenge the main characteristics of writing systems: the most fundamental challenge is the fact that the existence of ligatures makes the concept of letter problematic. Letters are graphemes out of which the alphabet is built, but they are valid only as abstract perception. As soon as we deal with material realization of writing, the concept of letter loses its validity: the occurrence of ligatures falsifies the definition of letters as what is separated by gaps inside a word."

24. In fact, many of them do appear in Unicode, but only for reasons of compatibility with legacy encodings, and their use is discouraged.

Contrary to the Latin script, the members of which are usually represented by separate graphs, the Arabic and the Syriac script have two levels of interaction between graphs:

1. on the primary level, a 4-form graph²⁵ is necessarily connected with the graph following it²⁶. Connecting strokes are horizontal and always occur at the baseline, as in |جج|;
2. on a secondary level, discretionary esthetic ligatures occur. In this case graphs are assembled vertically or diagonally, as in |جج| (Haralambous, 1994a).

Since there are two distinct levels of interaction between graphs, one may want to interfere on the first level (separate two graphs that are normally connected) or on the second level (avoid the use of a ligature and return to the standard pair of connected graphs). To allow this two-level interaction, Unicode recommends the use of two distinct characters:

1. the ZERO WIDTH NON-JOINER character (already mentioned above) that acts on the first level and separates graphs by changing their contextual form (the first graph will turn from initial to isolated and from medial to final, the second graph will turn from medial to initial and from final to isolated);
2. the ZERO WIDTH JOINER character that acts on the second level, by preventing any esthetic ligature but keeping the mandatory connection (and therefore not changing the graphs' contextual forms).

As an example, compare the following three:

- standard: جج ;
- with ZERO WIDTH JOINER: ججج ;
- with ZERO WIDTH NON-JOINER: جج·

Contextual form is a graphetic property of Arabic, but in some cases it can contribute to meaning production and can change the status of a grapheme from phonographic to logographic. For example, |ه| (the initial form of grapheme <ه>) is often used as the abbreviation of سنة هجرية "year of the Hegira," and is therefore a logogram. It can also have other meanings: for example, in the French-Arabic dictionary (*Mounged de poche français-arabe* 1991), several abbreviations are written in initial form: |م| for feminine gender (مؤمع), |ج| for plural number (جمع), |ه| for pronouns with nonhuman referents, and the same letter in isolated form |ه| for

25. In the Arabic writing system, graphs |و ز ر ذ د| are 2-form graphs (isolated and final form), graph |ء| has only one contextual form and all other graphs are 4-form graphs (isolated, initial, medial and final form).

26. Except for experimental versions of the Arabic script like those described in Haralambous (1998).

pronouns with human referents. Notice that no abbreviation dot is used so that their contextual form is the only indicator of their abbreviative nature.

Transgressing contextual rules for Arabic (or Syriac) graphs is part of the function of the ZERO WIDTH NON-JOINER character: to obtain the (initial) abbreviation |ا| through the M-H process, the ARABIC LETTER HEH character must be followed by the ZERO WIDTH NON-JOINER character. As this operation changes the nature of the grapheme into a logograph with a specific meaning given by the context, the ZERO WIDTH NON-JOINER is necessary also in the frame of the M-M process, even though the machine does not need to visualize Arabic in order to process it.

6. Emojis

Emojis are described in the Unicode Technical Report (Davis, 2019e). The word “emoji” comes from the Japanese 絵文字 (“e” = “picture” and “moji” = “written character”). Emojis are defined in Davis (ibid.) as

A colorful pictogram that can be used inline in text. Internally the representation is either (a) an image, (b) an encoded character, or (c) a sequence of encoded characters. (ibid., §1.4)

As many emojis are depicting humans, soon after their introduction questions began to arise about equal depiction of genders, ethnicities, religious minorities, etc. In 2016, the feminine brand *Always* started an advertisement campaign showing young women discussing gender representation in emojis, with slogans such “There aren’t enough emojis to show what girls can do”. To this the then First Lady Michelle Obama replied, on Women’s Day, March 8th, by a tweet:

Hey @Always! We would love to see a girl studying emoji. Education empowers girls around the world. #LetGirlsLearn #LikeAGirl

Following this presidential encouragement to emojis creators and smartphone manufacturers (see Stewart, Maria 2016), the Unicode Consortium faced the problem of sudden emoji multiplication: retaining only masculine white-skin forms was not politically correct, requiring a fixed number of variants for each emoji was unfeasible because of the risk of combinatorial explosion and because whatever the size of the set of variants, it had strong chances of ending up being incomplete in the long run.

The Unicode Consortium adopted a structuralist approach by gradually introducing *dimensions* in the set of emoji variants:

1. the *type of presentation* (typographic B&W or pictorial color);



FIGURE 6. Two ways of rendering emojis: “emoji presentation” on the right, and “text presentation” on the left.

2. *gender*;
3. *color of skin*;
4. *color of hair*;
5. as in Egyptian hieroglyphics, emojis sometimes picture humans or animals *sidewise*. According to cultural conventions (and, in particular, to direction of the dominant writing direction in a given culture), picturing a human or an animal facing/moving to the left or to the right do not have the same connotation, so a fifth dimension has been added: *direction* of sidewise presented human or animal.

To position an emoji in this 5-dimensional space, Unicode provides the mechanism of *emoji sequences*. As with combining sequences, the writer adds, after the “emoji base character” additional Unicode characters corresponding to the intended transformations; rendering engines then, after loading the font, select the appropriate emoji glyph whenever this is possible, or use a fallback mechanism when there is no glyph precisely fulfilling the writer’s demand.

There are five mechanisms allowing to obtain emoji variant glyphs:

1. *presentation sequences*, where a given emoji is followed by VARIATION SELECTOR-15 in order to be presented in B&W typographical style, or followed by VARIATION SELECTOR-16 in order to be presented in colorfull emoji style (see Fig. 6);
2. *modifier sequences*, where an emoji containing some part of human skin is followed by a character that will set the skin color, in five steps from light to dark:

Default image: 🍌; skin color 1: 🍌🏻; 2: 🍌🏼; 3: 🍌🏽; 4: 🍌🏾 and 5: 🍌🏿.

Unicode recommends that the default image (without modifier) should use “a generic, *non-realistic* skin tone” (usually: yellow²⁷);

3. *ZWJ sequences*, where some emojis are connected²⁸ by the ZERO WIDTH JOINER character, as in ligatures. The result of the ZWJ-joining of emojis is implementation dependent: it should result in the rendering of a single emoji incorporating visual elements from all joined emojis.

A recommended use of ZWJ sequences is to have gender appear explicitly in the emoji. For that, there are two mechanisms. Either an emoji depicting a person in a specific role is followed by a ZWJ character and then FEMALE SIGN ♀ or MALE SIGN ♂, or an object is preceded by the emoji MAN or WOMAN and the ZWJ character. Preceding the base emoji by the ADULT 🧑 emoji instead of MAN or WOMAN will produce a gender-neutral appearance. Here is an example: 🧑 is a male worker, this emoji followed by ♂ will remain as is, and followed by ♀ will become 🧑♀.

As can be seen in Fig. 7, in the specific case of the Apple Color Emoji font, the “gender-neutral” ADULT emoji is a morphed intermediate version between MAN and WOMAN, bearing anatomic characteristics and social conventions of both and (according to Western social conventions) having neither a moustache like MAN nor dyed lips like WOMAN.

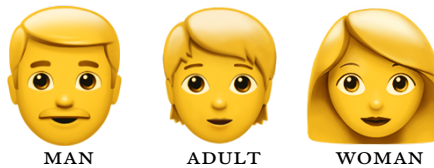


FIGURE 7. A closer look to the “generic” MAN, WOMAN and gender-neutral ADULT emojis, in the Apple Color Emoji font

In a similar manner, one can modify hair color of a face emoji by using ZWJ sequences with emoji components RED-HAIRED, CURLY-HAIRED, WHITE-HAIRED and BALD. Notice that brown/black hair is

27. It can be debated whether *non-realistic yellow* skin color is indeed politically correct, especially when it is combined with blond hair as in the example displayed.

28. While these mechanisms theoretically allow a very wide range of combinations, the Unicode Consortium also publishes Web pages and data files that list the combinations that implementers (font designers and keyboard designers) are expected to support, e.g., <http://www.unicode.org/emoji/charts/full-emoji-list.html#1f46d>. For example, there is an expectation that for a ‘person in lotus position,’ there is also a man and a woman in lotus position, whereas for a ‘person taking bath,’ there is no expectation for gender-specific variants.

default for face emojis, and therefore needs no extra component. Here are the effects of these modifications to the 🧑 emoji: red-haired 🧑🇷🇺, curly-haired 🧑🇨🇷, white-haired 🧑🇵🇪, and bald 🧑🇰🇪.

Finally one can indicate facing direction for emojis displaying humans, by using ZWJ sequences with arrow emojis.

The recommended order of components in an ZWJ sequence is the following: (1) base, (2) emoji modifier or presentation selector, (3) hair component, (4) gender component, (5) direction indicator.

4. A *flag sequence* is a special emoji displaying a black flag 🚩, followed by two ASCII characters representing a country (as in Davis (2019d)). One then obtains the flag of the country, for example 🚩fr should produce 🇫🇷;
5. a *tag sequence* is again a mechanism for obtaining flags, but this time for specific parts of countries, for example Wales as part of the United Kingdom. The approach is different: instead of having the black flag emoji character followed by exactly two ASCII characters, one writes an arbitrary number of “tag characters,”²⁹ and closes the sequence by a specific tag character called CANCEL TAG. The only constraint is that the total length of the sequence (including the black flag and the cancel tag) must be less or equal to 32. So, for example, to obtain the flag of Wales 🇨🇾 one will write 🚩 followed by tag characters gbwls and the CANCEL TAG character;

In linguistic terms, most emojis are pictograms; the exceptions to this rule are mostly cases where symbols are represented, such as 🙏, 🚫, 🚫, etc. In Peirce’s semiotics they are *iconic signs* since they physically resemble their real-world referents. Emojis are included in text, either in an adjunctive or in a substitutive way, and thus contribute to meaning production. Most of them are inherently ambiguous: a smiling face emoji 😊 can mean “I’m happy” or “don’t take it seriously, It’s just a joke” and many other interpretations according to the context. This playful and sometimes poetic ambiguity has certainly contributed to their popularity.

As all Unicode characters, emojis have names such as GRINNING FACE 😊, ROCKET 🚀 or ZOMBIE 🧟. Nevertheless, users are not necessarily aware of names: they choose emojis only according to their shape, and thus attach their own meaning to each emoji. During the act of communication these choices are then confronted to similar choices by other

29. “Tag characters” are ASCII characters transposed to the E0000₁₆ area, in the following sense: if the code point of LATIN SMALL LETTER A is hexadecimal 61₁₆ (that is decimal 97), then the code point of the corresponding TAG LATIN SMALL LETTER A character is E0000₁₆ + 61₁₆ = E0061₁₆ (decimal 917,601). In this text we will represent tag characters by underlined typewriter glyphs to prevent confusion with ordinary ASCII characters.

people, and this process results in a series of constantly evolving consensuses. In addition to that, on every new smartphone system release, a few hundred new emojis are added, enlarging the semantic spectrum available to users. C. Servais and V. Servais (2009) claim that “misunderstanding is the basic pattern of communication,” this is even more true when we consider communication by emojis.

To cut short this situation of ambient ambiguity and to solve once and for all the problem of emoji proliferation, the President of the Unicode Consortium, Mark Davis, submitted a groundbreaking proposal for indefinitely extending the number of emojis while precisely pinpointing their semantics.

The QID Emoji Proposal

The proposal (Davis, 2019a) was submitted to the Unicode Technical Committee on May 2nd, 2019, and at the time of writing of this paper it is not known whether it will be accepted.

Before describing the proposal, let us introduce the *Wikidata Project*. Wikidata is a collaborative knowledge base. It was launched by the Wikimedia Foundation in October 2012.

Wikidata has a graph³⁰ structure with *items*, *literals* and *media files* as vertices, and *statements* as edges. Items can be topics, concepts or objects. Statements connect items between them, items with literals (character strings or numbers), or items with media files. Each statement is an instance of a *property*. Each item has an identifier: the letter “Q” followed by a number; each property has an identifier: the letter “P” followed by a number. Statements may have *qualifiers* which are additional pieces of information. As of today (October 16th, 2019) Wikidata contains 63,573,864 data items and 6,762 properties.

As an example, the city of Brest (located in Brittany, France) is represented by item Q12193. Here are some of its statements:

Property	Value
P31 (instance of)	Q484170 (commune de France)
P31 (instance of)	Q1549591 (big city)
P17 (country)	Q142 (France)
P1313 (office held by head of government)	Q62266917 (Mayor of Brest)
P6 (head of government)	Q3084338 (François Cuillandre)


30. In this section the term “graph” refers to the mathematical structure (a set of binary relations) and *not to* the elementary material unit of writing, as in the rest of the paper.

By following these links we find out that the item “Brest” represents a big city in France, governed by a “Mayor Of Brest,” and this position is occupied by the referent of item Q3084338, called “François Cuillandre”. These are only 5 among the 136 statements provided for the item “Brest” in Wikidata.

Wikidata follows an intensional approach to information: items of the real world are entirely represented by their properties. These properties link items with other items, building a graph of relations between them. A human can retrieve information by following the relations of this graph and an inference engine can reply to queries formulated by humans.

Let us now describe the QID Emoji Proposal: Mark Davis proposes the establishment of a *one-to-one correspondence between emojis and Wikidata items*³¹. On a technical level, every emoji would be identified by a new kind of tag sequence, starting with a special generic emoji EMOJI TAG BASE, followed by a Wikidata QID identifier written in tag characters, and finally a CANCEL TAG character. For example, an emoji for the town of Brest would be obtained by the tag sequence [EMOJI TAG BASE]Q12193[CANCEL TAG].

The consequences of this initiative, if it is adopted, can be important. On the technical level, the size of the set of Unicode-encoded emojis will go from a few thousands to more than 60 millions. Smartphone providers will need to invent new ways of sharing fonts on the Web to provide emoji glyphs to any user requesting them—and for emojis not yet drawn, fallback glyph selection algorithms have to be applied.

But the most important consequence will be on the level of human communication: the new kind of emojis will be *significantly less ambiguous than written text*. For example, the textual sentence “I live in Brest” is ambiguous since there are at least 9 towns or villages with that name in the world (in Belarus, Bulgaria, Croatia, Czech Republic, France, Germany, Poland, Serbia and Slovenia), but the sentence “I live in [Q12193 emoji]” (potentially displayed as “I live in ”) is unambiguous³².

Furthermore we will witness a progressive shift from process M-H to process M-M: in process M-M the machine identifies concepts in linguistic data and replaces them with, for example, Wikidata identifiers. Using QID emojis in the M-H process, one will get a visual result similar to the existing one, but the Unicode data used to obtain it will already

31. With the possible exception of existing emojis, for which we don’t know whether they will be assigned to QIDs.

32. It is the code “Q12193” which is unambiguous, not the image of the emoji, where we see a tower that probably only an inhabitant or native of Brest will recognize as being the Tour Tanguy.

contain the necessary information for the NLP algorithm to unambiguously identify the meaning of the emoji that is part of the text.

For the moment, Wikidata items are only nouns, but one may very well imagine an extension to verbs (similar to WordNet, which has sections for nouns, verbs, adjectives and adverbs). This would allow the replacement of the verb “I live” by an “extended-QID” emoji, so that all lexical morphemes of the sentence “I live in Brest” are replaced by references to Wikidata. This process, known as *semantic annotation*, is very common in Artificial Intelligence.

Considering QID emojis with a large amount of optimism, one could say that, thanks to them, semantic annotation will become part of everyday human communication, and this may very well result in being a major turning point in human communication. But in fact, our optimism is limited since QID emojis could also create a range of problems and misunderstandings:

1. QIDs imply well-curated semantics, but emojis may quickly be repurposed. As an example, the peach emoji was already overloaded with meaning beyond that of the literal fruit. But in fall 2019, in a matter of weeks if not days, it acquired an additional meaning of “impeach,” based on the sudden prominence of the political topic in the USA and the phonetic similarity. Any hope of keeping the meaning of any emoji in any way limited to that of the underlying QID seems totally hopeless from the very start. There is no “emoji police,” and writers use emoji based on appearance, imagination, and consensus, rather than based on name or formal definition.
2. As a consequence of the previous point, it would be impossible for NLP software to put too much confidence in any kind of QID being used in an emoji. In many cases, somewhat paradoxically, deriving semantics from words (such as “peach”) might be considerably easier than deriving semantics from an emoji with a QID.
3. Although this may seem implied by the use of a QID grounded in ontology, there is no guarantee that a particular such emoji would be recognized as a depiction of the intended signifier. As an example, even the most prominent building or monument standing for (French) Brest may not be known to a wide range of people, even if these people have no problem to quickly identify Brest as a city in France.
4. While emojis are not specified to a single design, for many of them, the design is informed by the proposals made during the approval process and by the files depicting the newly accepted emojis. Major changes in interpretation, such as when the design of the pistol emoji was changed from a handgun to a water pistol (ABC News, 2018), happen only rarely. With QID emojis, if two people independently create emojis for the same QID, there is no guarantee that there is any kind of image similarity between the two emojis.

5. QID emojis may give the impression that literally everybody can start to use an emoji for any kind of concept. But experience with encoding existing minor scripts has shown that it is very difficult to make sure that the necessary fonts are widely available, even for well-defined language communities. And “install this font to read this Webpage” is a more realistic request than “install this font to view this emoji”. So realistically, QID emojis can only be introduced by major vendors, i.e., the groups that currently publish emoji fonts.
6. Emoji demand exists not only for well-defined ontological concepts, but also for combinations of concepts (e.g., cat with smiling face and tears). Such emojis cannot be created using QIDs, unless Wikidata gets diluted with such combinations.
7. Because each tag character needs four bytes for encoding, whereas ASCII characters need only one byte, it can easily be more efficient to use markup to add ontologically grounded meaning to text (including emojis) than to combine the meaning layer and the appearance in a single code. Using markup to add meaning also leads to a clear separation of concerns and a general solution (because it works for all kinds of text, not only a subset of emojis).

Conclusion

As an encoding, Unicode is a pervasive technology which probably will continue to exist as long as humanity will use text, be it in material or in disembodied form. But Unicode also provides a framework for the descriptive analysis of writing systems, which deserves to be scrutinized from a linguistic point of view, and this is what we attempted. We hope that this will be the starting point for research that will bring the community of Unicode aficionados and the community of (grapho)linguists closer together, and will result in a better understanding of the rationale of this wonderful human achievement.

References

- ABC News (2018). “Gun Emoji Replaced with Toy Water Pistol across All Major Platforms”. <https://perma.cc/V8DQ-ANKN>.
- André, Jacques and B. Borghi (1990). “Dynamic Fonts”. In: *PostScript Language Journal* 2.3, pp. 4–6.
- Anis, Jacques (1988). *L’écriture, théorie et descriptions*. Bruxelles: De Boeck.
- Bao, Jie et al. (2012). “OWL 2 Web Ontology Language Document Overview”. <https://www.w3.org/TR/owl2-overview/>.

- Bayar, Abdelouahad and Khalid Sami (2010). "Towards a Dynamic Font Respecting the Arabic Calligraphy". In: *Handbook of Research on E-services in the Public Sector: E-government Strategies and Advancements*. Ed. by Abid Thyab Al Ajeeli and Yousif A. Latif Al-Bastaki. Hershey PA: IGI Global, pp. 359–379.
- Bellamy-Royds, Amelia et al. (2018). "Scalable Vector Graphics (SVG) 2". <https://www.w3.org/TR/SVG2/>.
- Berglund, Anders (2006). "Extensible Stylesheet Language (XSL) Version 1.1". <https://www.w3.org/TR/xs111/>.
- Bray, Tim et al. (2008). "Extensible Markup Language (XML) 1.0". <https://www.w3.org/TR/xml/>.
- Brown, Keith and Jim Miller (2013). *The Cambridge Dictionary of Linguistics*. Cambridge: Cambridge University Press.
- Carlisle, David, Patrick Ion, and Robert Miner (2014). "Mathematical Markup Language (MathML) Version 3.0". <https://www.w3.org/TR/MathML3/>.
- Chomsky, Noam and M. Halle (1968). *The Sound Pattern of English*. New York: Harper & Row.
- Daniels, Peter and William Bright (1996). *The World's Writing Systems*. 2nd ed. Oxford: Oxford University Press.
- Davis, Mark (2019a). "QID Emoji Proposal". <http://www.unicode.org/L2/L2019/19082r-qid-emoji.pdf>.
- (2019b). "Unicode Standard Annex 29. Unicode Text Segmentation". <https://www.unicode.org/reports/tr29/>.
- (2019c). "Unicode Standard Annex 9. Unicode Bidirectional Algorithm". <https://www.unicode.org/reports/tr9/>.
- (2019d). "Unicode Technical Standard 35. Unicode Locale Data Markup Language". <https://www.unicode.org/reports/tr35/>.
- (2019e). "Unicode Technical Standard 51. Unicode Emoji". <http://www.unicode.org/reports/tr51/>.
- Dichy, Joseph (in this volume). "On the Writing System of Arabic. The Semiographic Principle as Reflected in Nashī Letter Shapes".
- Dukes, Kais, Eric Atwell, and Nizar Habash (2013). "Supervised Collaboration for Syntactic Annotation of Quranic Arabic". In: *Language Resources and Evaluation* 47.1, pp. 33–62.
- Dürscheid, Christa (2016). *Einführung in die Schriftlinguistik*. 5th ed. Göttingen: Vandenhoeck & Ruprecht.
- (2018). "Bild, Schrift, Unicode". In: *Sprache – Mensch – Maschine. Beiträge zu Sprache und Sprachwissenschaft, Computerlinguistik und Informationstechnologie für Jürgen Rolsboven aus Anlass seines sechsundsechzigsten Geburtstages*. Ed. by Guido Mensching et al. Köln: Kölner UniversitätsPublikationsServer, pp. 269–285.
- Dürst, Martin and Asmus Freytag (2000). "Unicode in XML and Other Markup Languages". <https://www.w3.org/TR/2000/NOTE-unicode-xml-20001215/>.

- Dürst, Martin and M. Suignard (2005). “Internationalized Resource Identifiers (IRIs)”. Request for Comments 3987.
- Gompel, Maarten van and Martin Reynaert (2013). “FoLiA: A Practical XML Format for Linguistic Annotation—a Descriptive and Comparative Study”. In: *Computational Linguistics in the Netherlands Journal* 3, pp. 63–81.
- Grzybek, Peter and Milan Rusko (2009). “Letter, Grapheme and (Allo-)Phone Frequencies: The Case of Slovak”. In: *Glottology* 2, pp. 30–48.
- Günther, Hartmut (1988). *Schriftliche Sprache: Strukturen geschriebener Wörter und ihre Verarbeitung*. Tübingen: Niemeyer.
- Haralambous, Yannis (1991). “Typesetting Old German: Fraktur, Schwabacher, Gotisch and Initials”. In: *TUGboat* 12.1, pp. 129–138.
- (1994a). “The Traditional Arabic Typecase Extended to the Unicode Set of Glyphs”. In: *Electronic Publishing—Origination, Dissemination, and Design* 8.2/3, pp. 125–138.
- (1994b). “Typesetting Khmer”. In: *Electronic Publishing—Origination, Dissemination, and Design* 7.4, pp. 197–215.
- (1995). “Tour du monde des ligatures”. In: *Cahiers GUTENBERG* 22, pp. 69–70.
- (1998). “Simplification of the Arabic Script: Two Different Approaches and Their Implementations”. In: *Springer Lecture Notes in Computer Science*. Vol. 1375: *Electronic Publishing, Artistic Imaging, and Digital Typography*, pp. 138–156.
- (2007). *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O’Reilly.
- (2013). “New Perspectives in Sinographic Language Processing Through the Use of Character Structure”. In: *Springer Lecture Notes in Computer Science*. Vol. 7816: *CICLing 2013: 14th International Conference on Intelligent Text Processing and Computational Linguistics, Samos*, pp. 201–217.
- Hayes, Patrick J. and Peter F. Patel-Schneider (2014). “RDF 1.1 Semantics”. <https://www.w3.org/TR/rdf11-mt/>.
- Hellwig, Oliver (2010–2019). “DCS—The Digital Corpus of Sanskrit”. <http://www.sanskrit-linguistics.org/dcs/>.
- Heninger, Andy (2019). “Unicode Standard Annex 14. Unicode Line Breaking Algorithm”. <https://www.unicode.org/reports/tr14/>.
- Kaplan, Ronald M. and Martin Kay (1994). “Regular Models of Phonological Rule Systems”. In: *Computational Linguistics* 29, pp. 331–378.
- Kazuo, Inamori [稲盛和夫] (2019). 心 [The Mind]. Tokyo: サンマーク出版 [Sunmark Publishing].
- Lotman, Jurij (1977). *Michigan Slavic Contributions*. Vol. 7: *The Structure of the Artistic Text*. Ann Arbor: The University of Michigan.
- Mackenzie, Charles E. (1980). *Coded Character Sets, History and Development*. Reading, MA: Addison-Wesley.

- Marneffe, Marie-Catherine de et al. (2013). “More Constructions, More Genres: Extending Stanford Dependencies”. In: *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*. Prague: Matfyzpress, pp. 187–196.
- Martinet, André (1970). “La double articulation du langage”. In: *La linguistique synchronique*. Paris: PUF, pp. 7–41.
- Meletis, Dimitrios (2015). *Graphetik. Form und Materialität von Schrift*. Glückstadt: Verlag Werner Hülsbusch.
- (2019). “Naturalness in scripts and writing systems: Outlining a Natural Grapholinguistics”. PhD thesis. University of Graz.
- Mounged de poche français-arabe* (1991). Beyrouth: Dar el-Machreq.
- Mousavi Jazayeri, S.M.V., Perette E. Michelli, and Sadd D. Abulhab (2017). *A Handbook of Early Arabic Kufic Script*. New York: Blautopf Publishing.
- Myers, James (2019). *The Grammar of Chinese Characters. Productive Knowledge of Formal Patterns in an Orthographic System*. London, New York: Routledge.
- Nehrlich, Thomas (2012). “Phänomenologie der Ligatur. Theorie und Praxis eines Schriftelements zwischen Letter und Lücke”. In: *Von Lettern und Lücken: zur Ordnung der Schrift im Bleisatz*. Ed. by Mareike Giertier and Rea Köppel. Paderborn: Wilhelm Fink, pp. 13–38.
- Pemperton, Steven et al. (2018). “XHTML 1.0 The Extensible HyperText Markup Language”. <https://www.w3.org/TR/xhtml1>.
- Rezec, Oliver (2009). “Zur Struktur des deutschen Schriftsystems. Warum das Graphem nicht drei Funktionen gleichzeitig haben kann, warum ein <a> kein <a> ist und andere Konstruktionsfehler des etablierten Beschreibungsmodells. Ein Verbesserungsvorschlag”. PhD thesis. Ludwig-Maximilians-Universität Munich.
- Sawicki, Marcin et al. (2001). “Ruby Annotation”. <https://www.w3.org/TR/2001/REC-ruby-20010531/>.
- Schopp, Jürgen F. (2008). “In Gutenbergs Fußstapfen: Translatio typographica. Zum Verhältnis von Typografie und Translation”. In: *Meta* 53, pp. 167–183.
- Servais, Christine and Véronique Servais (2009). “Le malentendu comme structure de la communication”. In: *Questions de communication* 15, pp. 21–49.
- Sproat, Richard (2000). *A Computational Theory of Writing Systems*. Cambridge: Cambridge University Press.
- Stewart, Maria (2016). “Michelle Obama Just Suggested a New Emoji to Empower Girls”. https://www.huffpost.com/entry/michelle-obama-always-emoji_n_56df3feee4b03a40567a78c3?guccounter=1.
- Stöckl, Harmut (2004). “Typographie: Gewand und Körper des Textes – Linguistische Überlegungen zu typographischer Gestaltung”. In: *Zeitschrift für angewandte Linguistik* 41, pp. 5–48.

- The Unicode Standard. Version 12.0—Core Specification* (2019). Mountain View, CA: The Unicode Consortium.
- Wmffre, Iwan (2008). *Contemporary Studies in Descriptive Linguistics*. Vol. 23: *Breton Orthographies and Dialects: The Twentieth-Century Orthography War in Brittany*. Bern: Peter Lang.