The Unicode Cookbook for Linguists

Managing writing systems using orthography profiles

Steven Moran & Michael Cysouw

Change dedication in localmetadata.tex

Contents

1 Writing Systems

1.1 Introduction

Writing systems arise and develop in a complex mixture of cultural, technological and practical pressures. They tend to be highly conservative, in that people who have learned to read and write in a specific way—however impractical or tedious—are mostly unwilling to change their habits. Writers tend to resist spelling reforms. In all literate societies there exists a strong socio-political mainstream that tries to force unification of writing (for example by strongly enforcing "right" from "wrong" writing in schools). However, there is also a large community of users who take as many liberties in their writing as they can get away with.

For example, the writing of tone diacritics in Yoruba is often proclaimed to be the right way to write, although many users of Yoruba orthography seem to be perfectly fine with leaving them out. As pointed out by the proponents of the official rules, there are some homographs when leaving out the tone diacritics (Olúmúyìw 2013: 44). However, writing systems (and the languages they represent) are normally full of homophones, which is normally not a problem at all for speakers of the language. More importantly, writing is not just a purely functional tool, but just as importantly it is a mechanism to signal social affiliation. By showing that you *know the rules* of expressing yourself in writing, others will more easily accept you as a worthy participant in their group. And that just as well holds for obeying to the official rules when writing a job application, as for obeying to the informal rules when writing an SMS to classmates in school. The case of Yoruba writing is an exemplary case, as even after more than a century of efforts to standardize the writing systems, there is still a wide range of variation in daily use (Olúmúyìw 2013).

The sometimes cumbersome and sometimes illogical structure, and the enormous variability of existing writing systems is a fact of life scholars have to accept and should try to adapt to as good as possible. Our plea here is a proposal for a formalization to do exactly that.

When considering the worldwide linguistic diversity, including all lesser-studied and endangered languages, there exist numerous different orthographies using symbols from the same scripts. For example, there are hundreds of orthographies using Latin-based alphabetic scripts. All of these orthographies use the same symbols, but these symbols differ in meaning and usage throughout the various orthographies. To be able to computationally use and compare different orthographies, we need a way to specify all orthographic idiosyncrasies in a computer-readable format (a process called TAILORING in Unicode parlance). We call such specifications ORTHOGRAPHY PROFILES. Ideally, these specifications have to be integrated into so-called Unicode locale descriptions, though we will argue that in practice this is often not the most useful solution for the kind of problems arising in the daily practice of linguistics. Consequently, a central goal of this paper is to flesh out the linguistic challenges for locale descriptions, and work out suggestions to improve their structure for usage in a linguistic context. Conversely, we also aim to improve linguists' understanding and appreciation for the accomplishments of the Unicode Consortium in the development of the Unicode Standard.

The necessity to computationally use and compare different orthographies most forcefully arises in the context of language comparison. Concretely, in our current research our goal is to develop quantitative methods for language comparison and historical analysis in order to investigate worldwide linguistic variation and to model the historical and areal processes that underlie linguistic diversity, cf. Steiner, Stadler & Cysouw (2011); List (2012a,b); List & Moran (2013); Moran & Prokić (2013). In this work, it is crucial to be able to flexibly process across numerous resources with different orthographies. In many cases even different resources on the *same* language use different orthographic conventions. Another orthographic challenge that we encounter regularly in our linguistic practice is electronic resources on a particular language that claim to follow a specific orthographic convention (often a resource-specific convention), but on closer inspection such resources are almost always not consistently encoded. Thus a second goal of our orthography profiles is to allow for an easy specification of orthographic conventions, and use such profiles to check consistency and to report errors to be corrected.

A central step in our proposed solution to this problem is the tailored grapheme separation of strings of symbols, a process we call GRAPHEME TOKENIZATION. Basically, given some strings of symbols (e.g. morphemes, words, sentences) in a specific source, our first processing step is to specify how these strings have to be separated into graphemes, considering the specific orthographic conventions

used in a particular source document. Our experience is that such a graphemic tokenization can be performed without extensive in-depth knowledge about the phonetic and phonological details of the language in question. For example, the specification that <ou> is a grapheme of English is a much easier task than to specify what exactly the phonetic values of this grapheme are in any specific occurrence in English words. Grapheme separation is a task that can be performed relatively reliably and with limited availability of time and resources (compare, for example, the task of creating a complete phonetic or phonological normalization).

Although grapheme tokenization is only one part of the solution, it is an important and highly fruitful processing step. Given a grapheme tokenization, various subsequent tasks become easier, like (a) temporarily reducing the orthography in a processing pipeline, e.g. only distinguishing high versus low vowels; (b) normalizing orthographies across sources (often including temporary reduction of oppositions), e.g. specifying an (approximate) mapping to the International Phonetic Alphabet; (c) using co-occurrence statistics across different languages (or different sources in the same language) to estimate the probability of grapheme matches, e.g. with the goal to find regular sound changes between related languages or transliterations between different sources in the same language.

Before we deal with these proposals, in the first part of this paper (Sections 1.2 through 3) we give an extended introduction to the notion of encoding (Section 1.2) and writing systems, both from a linguistic perspective and from the perspective of the Unicode Consortium (Section **??**). We consider the Unicode Standard to be a breakthrough (and ongoing) development that fundamentally changed the way we look at writing systems, and we aim to provide here a slightly more in-depth survey of the many techniques that are available in the standard. A good appreciation for the solutions that the Unicode Standard also allows for a thorough understanding of possible pitfalls that one might encounter when using it (Section 2). As an example of the current state-of-the-art, we discuss the rather problematic marriage of the International Phonetic Alphabet (IPA) with the Unicode Standard (Section 3).

The second part of the paper (Sections 4 and 5) describes our proposals for how to deal with the Unicode Standard in the daily practice of (comparative) linguists. First, we discuss the challenges of characterizing a writing system. To solve these problems, we propose the notions of orthography profiles, closely related to Unicode locale descriptions (Section 4). Finally, we discuss practical issues with actual examples (Section 5). We provide reference implementation of our proposals in R and in Python, available as open-source libraries.

The following conventions are followed in this paper. All phonemic and phonetic representations are given in the International Phonetic Alphabet (IPA), unless noted otherwise (International Phonetic Association 2005). Standard conventions are used for distinguishing between graphemic < >, phonemic / / and phonetic [ ] representations. For character descriptions, we follow the notational conventions of the Unicode Standard (Unicode Consortium 2014). Character names are represented in small capital letters (e.g. LATIN SMALL LETTER SCHWA) and code points are expressed as U+*n* where *n* is a four to six digit hexadecimal number, e.g. U+0256, which can be rendered as the glyph <ɘ>.

1.2 Encoding

There are many in-depth histories of the origin and development of writing systems (e.g. Robinson 1995; Powell 2012), a story that we therefore will not repeat here. However, the history of turning writing into computer readable code is not so often told, so we decided to offer a short survey of the major developments of such encoding here.[1] This history turns out to be intimately related to the history of telegraphic communication.

Telegraphy

Writing systems have existed for roughly 6000 years, allowing people to exchange messages through time and space. Additionally, to quickly bridge large geographic distances, telegraphic systems of communication (from Greek *τῆλε γράφειν* 'distant writing') have a long and widespread history since ancient times. The most common telegraphic systems worldwide are so-called whistled languages (Meyer 2015), but also drumming languages (Meyer, Dentel & Seifart 2012) and signalling by smoke, fire, flags or even change in water levels through hydraulic pressure have been used as forms of telegraphy.

Telegraphy was reinvigorated in the end of the eighteenth century through the introduction of so-called semaphoric systems by Claude Chapelle to convey

[1] Because of the recent history as summarized in this section, we have used mostly rather ephemeral internet sources. When not references by traditional literature in the bibliography, we have used http://www.unicode.org/history/ and various Wikipedia pages for the information presented here. A useful survey of the historical development of the physical hardware of telegraphy and telecommunication is Huurdeman (2003). Most books that discuss the development of encoding of telegraphic communication focus of cryptography, e.g. Singh (1999), and forego the rather interesting story of "open", i.e. non-cryptographic, encoding that is related here.

messages over large distances. Originally, various specially designed contraptions were used to send messages. Today, descendants of these systems are still in limited use, for example utilizing flags or flashing lights. The "innovation" of those semaphoric systems was that all characters of the written language were replaced one-to-one by visual signals. Since then, all telegraphic systems have taken this principle, namely that any language to be transmitted first has to be turned into some orthographic system, which subsequently is encoded for transmission by the sender, and then turned back into orthographic representation at the receiver side.[2] This of course implies that the usefulness of any such telegraphic encoding completely depends on the sometimes rather haphazard structure of orthographic systems.

In the nineteenth century, electric telegraphy lead to a new approach in which written language characters were encoded by signals sent through a copper wire. Originally, BISIGNAL CODES were used, consisting of two different signals. For example, Carl Friedrich Gauss in 1833 used positive and negative current (Mania 2008: 282). More famous and influential, Samuel Morse in 1836 used long and short pulses. In those bisignal codes each character from the written language was encoded with a different number of signals (between one and five), so two different separators are needed: one between signals and one between characters. For example, in Morse-code there is a short pause between signals and a long pause between characters.[3]

Binary encoding

From those bisignal encodings, true BINARY CODES developed with a fixed length of signals per character. In such systems only a single separator between signals is needed, because the separation between characters can be established by counting until a fixed number of signals has passed.[4] In the context of electric telegraphy, such a binary code system was first established by Émile Baudot in

[2] Sound and video-based telecommunication of course takes a different approach by ignoring the written version of language and directly encode sound waves or light patterns.

[3] Actually, Morse-code also includes an extra long pause between words. Interestingly, it took a long time to consider the written word boundary—using white-space—as a bona-fide character that should simply be encoded with its own code point. This happened only with the revision of the Baudot-code (see below) by Donald Murray in 1901, in which he introduced a specific white-space code. This principle has been followed ever since.

[4] Of course, no explicit separator is needed at all when the timing of the signals is known, which is the principle used in all modern telecommunication systems. An important modern consideration is also how to know where to start counting when you did not catch the start of a message, something that is known in Unicode as SELF SYNCHRONIZATION.

1870, using a fixed combination of five signals for each written character.[5] There are $2^5 = 32$ possible combination when using five binary signals; an encoding today designated as 5-bit. These codes are sufficient for all Latin letters, but of course they do not suffice for all written symbols, including punctuation and digits. As a solution, the Baudot code uses a so-called "shift" character, which signifies that from that point onwards—until shifted back—a different encoding is used, allowing for yet another set of 32 codes. In effect, this means that the Baudot code, and the International Telegraph Alphabet (ITA) derived from it, had an extra bit of information, so the encoding is actually 6-bit (with $2^6 = 64$ different possible characters). For decades, this encoding was the standard for all telegraphy and it is still in limited use today.

To also allow for different uppercase and lowercase letters and for a large variety of control characters to be used in the newly developing technology of computers, the American Standards Association decided to propose a new 7-bit encoding in 1963 (with $2^7 = 128$ different possible characters), known as the American Standard Code for Information Interchange (ASCII), geared towards the encoding of English orthography. With the ascent of other orthographies in computer usage, the wish to encode further variation of Latin letters (like German <ß> or various letters with diacritics, like <è>) led the Digital Equipment Corporation to introduce an 8-bit Multinational Character Set (MCS, with $2^8 = 256$ different possible characters), first used with the introduction of the VT220 Terminal in 1983.

Because 256 characters were clearly not enough for the many different characters needed in the world's writing systems, the ISO/IEC 8859 standard in 1987 extended the MCS to include 16 different 8-bit code pages. For example, part 5 was used for Cyrillic characters, part 6 for Arabic, and part 7 for Greek.[6] This system almost immediately was understood to be insufficient and impractical, so various initiatives to extend and reorganize the encoding started in the 1980s. This led, for example, to various proprietary encodings from Microsoft (e.g. Windows

[5] True binary codes have a longer history, going at least back to the Baconian cipher devised by Francis Bacon in 1605. However, the proposal by Baudot was the quintessential proposal leading to all modern systems.

[6] In effect, because $16 = 2^4$, this means that ISO/IEC 8859 was actually an $8+4 = 12$-bit encoding, though with very many duplicates by design, namely all ASCII codes were repeated in each 8-bit code page. To be precise, ISO/IEC 8859 used the 7-bit ASCII as the basis for each code page, and defined 16 different 7-bit extensions, leading to $(1 + 16) \cdot 2^7 = 2,176$ possible characters. However, because of overlap and not-assigned codes points the actual number of symbols was much smaller.

Latin 1) and Apple (e.g. Mac OS Roman), which one still sometimes encounters today.

More wide-ranging, various people in the 1980s started to develop true international code sets. In the United States, a group of computer scientists formed the Unicode Consortium, proposing a 16-bit encoding in 1991 (with $2^{16} = 65,536$ different possible characters). At the same time in Europe, the International Organization for Standardization (ISO) was working on ISO 10646 to supplant the ISO/IEC 8859 standard. Their first draft of the Universal Character Set (UCS) in 1990 was 31-bit (with theoretically $2^{31} = 2,147,483,648$ possible characters, but because of some technical restrictions only 679,477,248 were allowed). Since 1991, the Unicode Consortium and the ISO jointly develop the unicode standard, or ISO/IEC 10646, leading to the current system including the original 16-bit Unicode proposal as the basic multilingual plane, and 16 additional planes of 16-bit for further extensions (with in total $(1+16) \cdot 2^{16} = 1,114,112$ possible characters). The most recent version of the Unicode Standard (currently at version number 7.0) was published in June 2014 and it defines 112,956 different characters (Unicode Consortium 2014).

In the next section we provide a very brief overview of the linguistic terminology concerning writing systems before turning to the slightly different computational terminology in the next section on the Unicode Standard.

1.3 Linguistic terminology

Linguistically speaking, a writing system is a symbolic system that uses visible or tactile signs to represent language in a systematic way. The term writing system has two mutually exclusive meanings. First, it may refer to the way a particular language is written. In this sense the term refers to the writing system of a particular language, as, for example, in *'the Serbian writing system uses two scripts: Latin and Cyrillic.'* Second, the term writing system may also refer to a type of symbolic system as used among the world's languages to represent the language, as, for example, in *'alphabetic writing system.'* In this latter sense the term refers to how scripts have been classified according to the way that they encode language, as in, for example, *'the Latin and Cyrillic scripts are both alphabetic writing systems.'* To avoid confusion, this second notion of writing system would more aptly have been called script system.

Writing systems

Focussing on the first sense, we distinguish two different kinds of writing systems used for a particular language, namely transcriptions and orthographies.

First, TRANSCRIPTION is a scientific procedure (and also the result of that procedure) for graphically representing the sounds of human speech at the phonetic level. It incorporates a set of unambiguous symbols to represent speech sounds, including conventions that specify how these symbols should be combined. A transcription system is a specific system of symbols and rules used for transcription of the sounds of a spoken language variety. In principle, a transcription system should be language-independent, in that it should be applicable to all spoken human languages. The INTERNATIONAL PHONETIC ALPHABET (IPA) is a commonly used transcription system that provides a medium for transcribing languages at the phonetic level. However, there is a long history of alternative kinds of transcription systems (see Kemp 2006) and today various alternatives are in widespread use (e.g. X-SAMPA and Cyrillic-based phonetic transcription systems). Many users of IPA do not follow the standard to the letter, and many dialects based on the IPA have emerged, e.g. the Africanist and Americanist transcription systems. Note that IPA symbols are also often used to represent language on a phonemic level. It is important to realize that in this usage the IPA symbols are not a transcription system, but rather an orthography (though with strong links to the pronunciation). Further, a transcription system does not need to be as highly detailed as the IPA. It can also be a system of broad sound classes. Although such an approximative transcription is not normally used in linguistics, it is widespread in technological approaches (Soundex and variants, e.g. Knuth 1973: 391–392; Postel 1969; Beider & Morse 2008), and it is sometimes fruitfully used in automatic approaches to historical linguistics (Dolgopolsky 1986; List 2012c; Brown, Holman & Wichmann 2013).

Second, an ORTHOGRAPHY specifies the symbols, punctuations, and the rules in which a specific language is written in a standardized way. Orthographies are often based on a phonemic analysis, but they almost always include idiosyncrasies because of historical developments (like sound changes or loans) and because of the widely-followed principle of lexical integrity (i.e. the attempt to write the same lexical root in a consistent way, also when synchronic phonemic rules change the pronunciation, as for example with final devoicing in many Germanic languages). All orthographies are language-specific (and often even resource-specific), although individual symbols or rules might be shared between languages. A PRACTICAL ORTHOGRAPHY is a strongly phoneme-based writing system designed for practical use by speakers. The mapping relation between phonemes

and graphemes in practical orthographies is purposely shallow, i.e. there is mostly a systematic and faithful mapping from a phoneme to a grapheme. Practical orthographies are intended to jumpstart written materials development by correlating a writing system with the sound units of a language (cf. Meinhof & Jones 1928). Symbols from the IPA are often used by linguists in the development of such practical orthographies for languages without writing systems, though this usage of IPA symbols should not be confused with transcription (as defined above).

Further, a TRANSLITERATION is a mapping between two different orthographies. It is the process of "recording the graphic symbols of one writing system in terms of the corresponding graphic symbols of a second writing system" (Kemp 2006: 396). In straightforward cases, such a transliteration is simply a matter of replacing one symbol with another. However, there are widespread complications, like one-to-many or many-to-many mappings, which are not always easy, or even possible, to solve without listing all cases individually (cf. Moran 2012: Ch. 2).

Script systems

Different kinds of writing systems are classified into script systems. A SCRIPT is a collection of distinct symbols as employed by one or more orthographies. For example, both Serbian and Russian are written with subsets of the Cyrillic script. A single language, like Serbian or Japanese, can also be written using orthographies based on different scripts. Over the years linguists have classified script systems in a variety of ways, with the tripartite classification of logographic, syllabic, and alphabetic remaining the most popular, even though there are at least half a dozen different types of script systems that can be distinguished (Daniels 1990, 1996).

Breaking it down further, a script consists of GRAPHEMES, and graphemes consist of CHARACTERS. In the linguistic terminology of writing systems, a CHARACTER is a general term for any self-contained element in a writing system.[7] Although in literate societies most people have a strong intuition about what the characters are in their particular orthography or orthographies, it turns out that the separation of an orthography into separate characters is far from trivial. The widespread intuitive notion of a character is strongly biased towards educational traditions, like the alphabet taught at schools, and technological possibilities, like the available type pieces in a printer's job case, the keys on a typewriter, or the

[7] There is a second interpretation of the term CHARACTER, i.e. a conventional term for a unit in the Chinese writing system (Daniels 1996). This interpretation will not be further explored in this paper.

symbols displayed in Microsoft Word's symbol browser. In practice, characters often consist of multiple building blocks, each of which could be considered a character in its own right. For example, although a Chinese character may be considered to be a single basic unanalyzable unit, at a more fine-grained level of analysis the internal structure of Chinese characters is often comprised of smaller semantic and phonetic units that should be considered characters (Sproat 2000). In alphabetic scripts, this problem is most forcefully exemplified by diacritics.

A DIACRITIC is a mark, or series of marks, that may be above, below, or through other characters (Gaultney 2002). Diacritics are sometimes used to distinguish homophonous words, but they are more often used to indicate a modified pronunciation (Daniels & Bright 1996: xli). The central question is whether, for example, <e>, <è>, <a> and <à> should be considered four characters, or different combinations of three characters. In general, multiple characters together can form another character, and it is not always possible to decide on principled grounds what should be the basic building blocks of an orthography.

For that reason, it is better to analyze an orthography as a collection of graphemes. A GRAPHEME is the basic, minimally distinctive symbol of a particular writing system, alike to the phoneme is an abstract representation of a distinct sound in a specific language. The term GRAPHEME was modeled after the term PHONEME and represents a contrastive graphical unit in a writing system (see Kohrt 1986 for a historical overview of the term grapheme). Most importantly, a single grapheme regularly consists of multiple characters, like <th>, <ou> and <gh> in English (note that each character in these graphemes is also a separate grapheme in English). Such complex graphemes are often used to represent single phonemes. So, a combination of characters is used to represent a single phoneme. Note that the opposite is also found in writing systems, in cases in which a single character represents a combination of two or more phonemes. For example, <x> in English orthography represents a combination of the phonemes /k/ and /s/.

Further, conditioned or free variants of a grapheme are called ALLOGRAPHS. For example, the distinctive forms of Greek sigma are conditioned, with <σ> being used word-internally and <ς> being used at the end of a word. In sum, there are many-to-many relationships between phonemes and graphemes as they are expressed in the myriad of language- and resource-specific orthographies.

This exposition of the linguistic terminology involved in describing writing systems has been purposely brief. We have highlighted some of the linguistic notions that are pertinent, yet sometimes confused with, the technological defini-

tions developed for the computational processing of the world's writing systems, which we describe in the next section.

1.4 The Unicode approach

The conceptualization and terminology of writing systems was rejuvenated by the development of the Unicode Standard, with major input from Mark Davis, co-founder and long-term president of the Unicode Consortium. For many years, "exotic" writing systems and phonetic transcription systems on personal computers were constrained by the American Standard Code for Information Interchange (ASCII) character encoding scheme, based on the Latin script, which only allowed for a strongly limited number of different symbols to be encoded. This implied that users could either use and adopt the (extended) Latin alphabet or they could assign new symbols to the small number of code points in the ASCII encoding scheme to be rendered by a specifically designed font (Bird & Simons 2003). In this situation, it was necessary to specify the font together with each document to ensure the rightful display of its content. To alleviate this problem of assigning different symbols to the same code points, in the late 80's and early 90's the Unicode Consortium set itself the ambitious goal of developing a single universal character encoding to provide a unique number, a code point, for every character in the world's writing systems. Nowadays, the Unicode Standard is the default encoding of the technologies that support the World Wide Web and for all modern operating systems, software and programming languages.

The Unicode Standard

The Unicode Standard represents a massive step forward because it aims to eradicate the distinction between universal (ASCII) versus language-particular (font) by adding as much as possible language-specific information into the universal standard. However, there are still language/resource-specific specifications necessary for the proper usage of Unicode, as will be discussed below. Within the Unicode structure many of these specifications can be captured by so-called LOCALE DESCRIPTIONS, so we are moving to a new distinction of universal (Unicode Standard) versus language-particular (locale description). The major gain is a much larger compatibility on the universal level (because Unicode standardizes a much larger portion of writing system diversity), and much better possibilities for automatic processing on the language-particular level (because locale descriptions are computer readable specifications).

Each version of the Unicode Standard (Unicode Consortium 2014, as of writing at version 7) consists of a set of specifications and guidelines that include (i) a core specification, (ii) code charts, (iii) standard annexes and (iv) a character database.[8] The CORE SPECIFICATION is a book directed toward human readers that describes the formal standard for encoding multilingual text. The CODE CHARTS provide a humanly readable online reference to the character contents of the Unicode Standard in the form of PDF files. The UNICODE STANDARD ANNEXES (UAX) are a set of technical standards that describe the implementation of the Unicode Standard for software development, Web standards, and programming languages. The UNICODE CHARACTER DATABASE (UCD) is a set of computer-readable text files that describe the character properties, including a set of rich character and writing system semantics, for each character in the Unicode Standard. In this section, we introduce the basic Unicode concepts, but we will leave out many details. Please consult the above mentioned full documentation for a more detailed discussion. Further note that the Unicode Standard is exactly that, namely a standard. It normatively describes notions and rules to be followed. In the actual practice of applying this standard in a computational setting, a specific implementation is necessary. The most widely used implementation of the Unicode Standard is the INTERNATIONAL COMPONENTS FOR UNICODE (ICU), which offers C/C++ and Java libraries implementing the Unicode Standard.[9]

Character encoding system

The Unicode Standard is a CHARACTER ENCODING SYSTEM which goal it is to support the interchange and processing of written characters and text in a computational setting. Underlyingly, the character encoding is represented by a range of numerical values called a CODE SPACE, which is used to encode a set of characters. A CODE POINT is a unique non-negative integer within a code space (i.e. within a certain numerical range). In the Unicode Standard character encoding system, an ABSTRACT CHARACTER, for example the LATIN SMALL LETTER P, is mapped to a particular code point, in this case the decimal value 112, normally represented in

[8] All documents of the Unicode Standard are available at http://www.unicode.org/versions/latest/. For a quick survey of the use of terminology inside the Unicode Standard, their glossary is particularly useful, available at http://www.unicode.org/glossary/. For a general introduction to the principles of Unicode, Chapter 2 of the core specification, called GENERAL STRUCTURE, is particularly insightful. Different from many other documents of the Unicode Standard, this general introduction is relatively easy to read and illustrated with many interesting examples from various orthography traditions all over the world.

[9] More information about the ICU is available here: http://icu-project.org.

hexadecimal, which then looks in Unicode parlance as U+0070.[10] That encoded abstract character is rendered on a computer screen (or printed page) as a GLYPH, e.g. <p>, depending on the FONT and the context in which that character appears.

In Unicode Standard terminology, an (abstract) CHARACTER is the basic encoding unit. The term CHARACTER can be quite confusing due to its alternative definitions across different scientific disciplines and because in general the word CHARACTER means many different things to different people. It is therefore often preferable to refer to Unicode characters simply as CODE POINTS, because there is a one-to-one mapping between Unicode characters and their numeric representation. In the Unicode approach, a character refers to the abstract meaning and/or general shape, rather than a specific shape, though in code tables some form of visual representation is essential for the reader's understanding. Unicode defines characters as abstractions of orthographic symbols, and it does not define visualizations for these characters (although it does presents examples). In contrast, a GLYPH is a concrete graphical representation of a character as it appears when rendered (or rasterized) and displayed on an electronic device or on printed paper. For example, <g *g* **g** g ***g*** **g**> are different glyphs of the same character, i.e. they may be rendered differently depending on the typography being used, but they all share the same code point. From the perspective of Unicode they are *the same thing*. In this approach, a FONT is then simply a collection of glyphs linked to code points. Allography is not specified in Unicode (expect for a few exceptional cases, due to legacy encoding issues), but can be specified in a font as a CONTEXTUAL VARIANT (a.k.a. presentation form).

Each code point in the Unicode Standard is associated with a set of CHARACTER PROPERTIES as defined by the Unicode character property model.[11] Basically, those properties are just a long list of values for each character. For example, code point U+0047 has the following properties (among many others):

- Name: LATIN CAPITAL LETTER G
- Alphabetic: YES
- Uppercase: YES

[10] Hexadecimal (base-16) 0070 is equivalent to decimal (base-10) 112, which can be calculated by considering that $(0 \cdot 16^3) + (0 \cdot 16^2) + (7 \cdot 16^1) + (0 \cdot 16^0) = 7 \cdot 16 = 112$. Underlyingly, computers will of course treat this code point binary (base-2) as 11100000, as can be seen by calculating that $(1 \cdot 2^7) + (1 \cdot 2^6) + (1 \cdot 2^5) + (0 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0) = 64 + 32 + 16 = 112$.

[11] The character property model is described in http://www.unicode.org/reports/tr23/, but the actual properties are described in http://www.unicode.org/reports/tr44/. A simplified overview of the properties is available at http://userguide.icu-project.org/strings/properties. The actual code tables listing all properties for all Unicode code points are available at http://www.unicode.org/Public/UCD/latest/ucd/.

- Script: LATIN
- Extender: NO
- Simple_Lowercase_Mapping: 0067

These properties contain the basic information of the Unicode Standard and they are necessary to define the correct behavior and conformance required for interoperability in and across different software implementations (as defined in the Unicode Standard Annexes). The character properties assigned to each code point is based on each character's behavior in the real-world writing traditions. For example, the corresponding lowercase character to U+0047 is U+0067 (though note that the relation between uppercase and lowercase is in many situations much more complex than this, and Unicode has further specifications for those cases). Another use of properties is to define the script of a character.[12] In practice, script is simply defined for each character as the explicit SCRIPT property in the Unicode Character Database.

One frequently referenced property is the BLOCK property, which is often used in software applications to impose some structure to the large number of Unicode characters. Each character in Unicode belongs to a specific block. These blocks are basically an organizational structure to alleviate the administrative burden of keeping Unicode up-to-date. Blocks consist of characters that in some way belong together, so that characters are easier to find. Some blocks are connected with a specific script, like the Hebrew block or the Gujarati block. However, blocks are predefined ranges of code points, and often there will come a point after which the range is completely filled. Any extra characters will have to be assigned somewhere else. There is, for example, a block ARABIC, which contains most Arabic symbols. However, there is also a block ARABIC SUPPLEMENT, ARABIC PRESENTATION FORMS-A and ARABIC PRESENTATION FORM-B. The situation with Latin symbols is even more extreme. In general, the names for block should not be taken as a definitional statement. For example, many IPA symbols are not located in the aptly-names block IPA EXTENSIONS, but in other blocks (see Section 3.3).

[12] The Unicode Glossary defines the term SCRIPT as a "collection of letters and other written signs used to represent textual information in one or more writing systems. For example, Russian is written with a subset of the Cyrillic script; Ukrainian is written with a different subset. The Japanese writing system uses several scripts."

Grapheme clusters

There are many cases in which a sequence of characters (i.e. a sequence of more than one code point) represents what a user perceives as an individual unit in a particular orthographic writing system. For this reason the Unicode Standard differentiates between ABSTRACT CHARACTER and USER-PERCEIVED CHARACTER. Sequences of multiple code points that correspond to a single user-perceived characters are called grapheme clusters in Unicode parlance. Grapheme clusters come in two flavors: (default) grapheme clusters and tailored grapheme clusters.

The (default) GRAPHEME CLUSTERS are locale-independent graphemes, i.e. they always apply when a particular combination of characters occurs independent of the writing system in which they are used. These character combinations are defined in the Unicode Standard as functioning as one TEXT ELEMENT.[13] The simplest example of a grapheme cluster is a base character followed by a letter modifier character. For example, the sequence <n> + <◌̃> (i.e. LATIN SMALL LETTER N at U+006E, followed by COMBINING TILDE at U+0303) combines visually into <ñ>, a user-perceived character in writing systems like that of Spanish. So, what the user perceives as a single character actually involves a multi-code-point sequence. Note that this specific sequence can also be represented with a single so-called PRECOMPOSED code point, LATIN SMALL LETTER N WITH TILDE at U+00F1, but this is not the case for all multi-code point character sequences. The problem that there multiple encodings possible for the same text element has been acknowledged early on in the Unicode Standard (e.g. for <ñ>, the sequence U+006E U+0303 should in all situations be treated identically to the precomposed U+00F1), and a system of CANONICAL EQUIVALENCE is available for such situations. Basically, the Unicode Standard offers different kind of normalizations to either decompose all precomposed characters (called NFD, NORMALIZATION FORM CANONICAL DECOMPOSITION), or compose as much as possible combinations (called NFC, NORMALIZATION FORM CANONICAL COMPOSITION). In current practice of software development, NFC seems to be preferred in most situations and is widely proposed as the preferred canonical form.

More difficult for text processing, because less standardized, is what the Unicode Standard terms TAILORED GRAPHEME CLUSTERS. Tailored grapheme clusters are locale-dependent graphemes, i.e. such combination of characters do not function as text elements in all situations. For example, the sequence <c> + <h> for the Slovak digraph <ch> or the sequence <ky> in the Sisaala practical orthog-

[13] The Unicode Glossary defines text element as: "A minimum unit of text in relation to a particular text process, in the context of a given writing system. In general, the mapping between text elements and code points is many-to-many."

raphy (pronounced as IPA /tʃ/, Moran 2006). These grapheme clusters are TAILORED in the sense that they must be specified on a language-by-language or writing-system-by-writing-system basis. The Unicode Standard provides technological specifications for creating locale specific data in so-called UNICODE LOCALE DESCRIPTIONS, i.e. a set of specification that defines a set of language-specific elements (e.g. tailored grapheme clusters, collation order, capitalization-equivalence), as well as other special information, like how to format numbers, dates, or currencies. Locale descriptions are saved in the COMMON LOCALE DATA REPOSITORY (CLDR),[14] a repository of language-specific definitions of writing system properties, each of which describes specific usages of characters. Each locale can be encoded in a document using the LOCALE DATA MARKUP LANGUAGE (LDML). LDML is an XML format and vocabulary for the exchange of structured locale data. Unicode Locale Descriptions allow users to define language- or even resource-specific writing systems or orthographies.[15] However, there are various drawbacks of locale descriptions for the daily practice of linguistic work in a multilingual setting.

[14] More information about the CLDR can be found here: http://cldr.unicode.org/.

[15] The Unicode Glossary defines WRITING SYSTEM only very loosely, as it is not a central concept in the Unicode Standard. A writing system is, "A set of rules for using one or more scripts to write a particular language. Examples include the American English writing system, the British English writing system, the French writing system, and the Japanese writing system."

2 Unicode pitfalls

2.1 Wrong it is not

In this chapter we describe some of the most common pitfalls that we have encountered when using the Unicode Standard in our own work, or in discussion with other linguists. This section is not meant as a criticism of the decisions made by the Unicode Consortium; on the contrary, we aim to highlight where the technological aspects of the Unicode Standard diverge from many users' intuitions. What have sometimes been referred to as problems or inconsistencies in the Unicode Standard are mostly due to legacy compatibility issues, which can lead to unexpected behavior by linguists using the standard. However, there are also some cases in which the Unicode Standard has made decisions that theoretically could have been taken differently, but for some reason or another (mostly very good reasons) were accepted as they are now. We call behavior that executes without error but does something different than the user expected—often unknowingly—a PITFALL.

In this context, it is important to realize that the Unicode Standard was not developed to solve linguistic problems per se, but to offer a consistent computational environment for written language. In those cases in which the Unicode Standard behaves differently as expected, we think it is important not to dismiss Unicode as "wrong" or "deficient", because our experience is that in almost all cases the behavior of the Unicode Standard has been particularly well thought through. The Unicode Consortium has a more wide-ranging view of matters and often examines important practical use-cases that from a linguistic point of view are normally not considered. Our general guideline for dealing with the Unicode Standard is to accept it as it is, and not to battle windmills. Alternatively, of course, it is possible to actively engage in the development of the standard itself, an effort that is highly appreciated by the Unicode Consortium.

2.2 Pitfall: Characters are not glyphs

A central principle of Unicode is the distinction between character and glyph. A character is the abstract notion of a symbol in a writing system, while a glyph is the concrete drawing of such a symbol. In practice, there is a complex interaction between characters and glyphs. A single Unicode character may of course be rendered as a single glyph. However, a character may also be a piece of a glyph, or vice-versa. Actually, all possible relations between glyphs and characters are attested.

First, a single character may have different contextually determined glyphs. For example, characters in writing systems like Hebrew and Arabic have different glyphs depending on where they appear in a word. Some letters in Hebrew change their form at the end of the word, and in Arabic, primary letters have four contextually-sensitive variants (isolated, word initial, medial and final). Second, a single character may be rendered as a sequence of multiple glyphs. For example, in Tamil one Unicode character may result in a combination of a consonant and vowel, which are rendered as two adjacent glyphs by fonts that supports Tamil. Third, a single glyph may be a combination of multiple characters. For example, the ligature <fi>, a single glyph, is the result of two characters, <f> and <i>, that have undergone glyph substitution by font rendering (see also Section 2.5). Like contextually-determined glyphs, ligatures are (intended) artifacts of text processing instructions. Finally, a single glyph may be a part of a character, as exemplified by diacritics.

Further, the rendering of a glyph is dependent on the font being used. For example, the Unicode character LATIN SMALL LETTER G appears as <g> and <ɡ> in the Linux Libertine and Courier fonts, respectively, because their typefaces are designed differently. Furthermore, font face may change the visual appearance of a character, for example Times New Roman two-story <a> changes to a single-story glyph in italics <*a*>. This becomes a real problem for some phonetic typesetting (see Section 3.4).

In sum, character-to-glyph mappings are complex technological issues that the Unicode Consortium has had to address in the development of the Unicode Standard, but for the lay user they can be utterly confusing because visual rendering does not (necessarily) indicate logical encoding.

2.3 Pitfall: Characters are not graphemes

The Unicode Standard is a character encoding system, and not a writing system encoding system. This most forcefully becomes clear with the notion of grapheme. From a linguistic point of view, graphemes are the basic building blocks of a writing system (see Section 1.3). It is extremely common, up to the point of being universally attested, that writing systems use combinations of multiple symbols as a single grapheme, like <sch>, <th> or <ei>. There is no possibility to encode such complex graphemes using the Unicode Standard.

The Unicode Standard deals with complex graphemes only inasmuch they consist of base characters with diacritics (see Section 3.10 for a discussion of the notion of diacritic). The Unicode Standard calls such combination "grapheme clusters". Complex graphemes consisting of multiple base characters, like <sch>, are called "tailored grapheme clusters" in Unicode parlance (see Section 1.4).

Inspecting the Unicode Standard, there appear to be special Unicode characters to "glue" together characters into larger tailored grapheme clusters, specifically the ZERO WIDTH JOINER at U+200D and the COMBINING GRAPHEME JOINER at U+034F. However, these characters are confusingly named (cf. Section 2.7). Both codepoints actually do not join characters, but explicitly separate them. The zero-width joiner (ZWJ) can be used to solve special problems related to ordering (called "collation" in Unicode parlance). The combining grapheme joiner (CGJ) can be used to separate characters that are not supposed to form ligatures.

To solve the issue of tailored grapheme clusters, Unicode offers some assistance in the form of the Unicode Locale Descriptions. However, in the practice of linguistic research, this is not a real solution. For that reason we propose to use orthography profiles (see Chapter 4). Basically, both orthography profiles and locale descriptions offer a way to specify tailored grapheme clusters. For example, for English one could specify that <sh> is such a cluster. Consequently, this sequence of characters is then always interpreted as a complex grapheme. For cases in which this is not the right decision, like in the English word *mishap*, the ZERO WIDTH JOINER at U+200D has to be entered between <s> and <h>.

2.4 Pitfall: Missing glyphs

The Unicode Standard is often praised (and deservedly so) for solving many of the perennial problems with the interchange and display of the world's writing systems. However, a common complaint from users is that, while the praise may be true, they mostly just see some boxes on their screen instead of those promised

symbols. The problem of course is that users' computers do not have any glyphs installed matching the Unicode code points in the file they are trying to inspect. It is important to realize that internally in the computer everything still works as expected: any handling of Unicode code points works independently of how they are displayed on the screen. So, although a user might only see boxes being displayed, this user should be assured that everything is still in order.

The central problem behind the missing glyphs is that designing actual glyphs includes a lot of different considerations and it is a time-consuming process. Many traditional expectations of how specific characters should look like have to be taken into account when designing glyphs. Those expectations are often not well documented, and it is mostly up to the knowledge and experience of the font designer to try and conform to them as good as possible. Therefore, most designers produce fonts only including glyphs for certain parts of the Unicode Standard, namely for those characters they feel comfortable with. At the same time, the number of characters defined by the Unicode Standard is growing with each new version, so it is neigh impossible for any designer to produce glyphs for all characters. The result of this is that, almost necessarily, each font only includes glyphs for a subset of the characters in the Unicode Standard.

The simple solution to missing glyphs is thus to install additional fonts providing additional glyphs. For the more exotic characters there is often not much choice. There are a few particularly large fonts that might be considered. First, there is the Everson Mono font made by Michael Everson, which currently includes 9,756 different glyphs (not including Chinese) updated up to Unicode 7.0.[1] Already a bit older is the Titus Cyberbit Basic font made by Jost Gippert and Carl-Martin Bunz, which includes 10,044 different glyphs (not including Chinese), but not including newer characters added after Unicode 4.0.[2]

Further, we suggest to always install at least one so-called fall-back font, which provides glyphs that at least show the user some information about the underlying encoded character. Apple Macintoshes have such a font (which is invisible to the user), which is designed by Michael Everson and made available for other systems through the Unicode Consortium.[3] Further, the GNU Unifont is a clever way to produce bitmaps approximating the intended glyph of each available character, updated to Unicode 7.0.[4] Finally, the Summer Institute of Linguistics provides a SIL Unicode BMP Fallback Font, currently available up

[1] Everson Mono is available as shareware at http://www.evertype.com/emono/.

[2] Titus Cyberbit Basic is available at http://titus.fkidg1.uni-frankfurt.de/unicode/tituut.asp.

[3] The Apple/Everson fallback font is available for non-Macintosh users at http://www.unicode.org/policies/lastresortfont_eula.html.

[4] The GNU Unifont is available at http://unifoundry.com/unifont.html.

to Unicode version 6.1. This font does not even attempt to show a real glyph, but only shows the hexadecimal code inside a box for each character, so a user can at least see the Unicode codepoint of the character to be displayed.[5]

2.5 Pitfall: Faulty rendering

A similar complaint to missing glyphs, discussed previously, is that while there might be a glyph being displayed, it does not look right. There are two reasons for unexpected visual display, namely automatic font substitution and faulty rendering. Like missing glyphs, any such problems are independent from the Unicode Standard. The Unicode Standard only includes very general information about characters and leaves the specific visual display to others to decide on. Any faulty display is thus not to be blamed on the Unicode Consortium, but on a complex interplay of different mechanisms happening in a computer to turn Unicode codepoints into visual symbols. We will only sketch a few aspects of this complex interplay here.

Most modern software applications (like Microsoft Word) offer some approach to AUTOMATIC FONT SUBSTITUTION. This means that when a text is written in a specific font (e.g. Times New Roman) and an inserted Unicode character does not have a glyph within this font, then the software application will automatically search for another font to display the glyph. The result will be that this specific glyph will look slightly different from the others. This mechanism works differently depending on the software application, and mostly only limited user influence is expected and little feedback is given, which might be rather frustrating to font-aware users.[6]

The other problem with visual display is related to the so-called FONT RENDERING. Font rendering refers to the process of the actual positioning of Unicode characters on a page of written text. This positioning is actually a highly complex problem, and many things can go wrong in the process. Well-known rendering problems, like proportional glyph size or ligatures are reasonably well understood. In contrast, the positioning of multiple diacritics relative to a base

[5] The SIL Unicode BMP Fallback Font is available at http://scripts.sil.org/UnicodeBMPFallbackFont.

[6] For example, Apple Pages does not give any feedback that a font is being replaced, and the user does not seem to have any influence on the choice of replacement (except by manually marking all occurrences). In contrast, Microsoft Word does indicate the font replacement by showing the name in the font menu of the font replacement. However, Word simply changes the font completely, so any text written after the replacement is written in a different font as before. Both behaviors leave much to be desired.

character is still a widespread problem, even within the Latin script. Especially when more than one diacritic is supposed to be placed above (or below) each other, this often leads to unexpected effects in many modern software applications. The problems arising in Arabic and in many southeast Asian scripts (like Devanagari or Burmese) are even more complex.

To understand where any problems arise it is important to realize that there are basically three different approaches to font rendering. The most widespread is Adobe's and Microsoft's OpenType system. This approach makes it relatively easy for font developers, as the font itself does not include all details about the precise placement of individual characters. For those details, additional script-descriptions are necessary. All of those systems can lead to unexpected behavior.[7] Alternative systems are Apple Advanced Typography (AAT) and the open-source Graphite system from the Summer Institute of Linguistics (SIL).[8] In both of these systems, a larger burden is placed on the description inside the font.

There is mostly no real solution to problems arising from faulty font rendering. Switching to another software application that offers better handling is the only real alternative, but this is normally not an option for daily work. The experience with rendering on the side of the software industry is developing quickly, so we can expect the situation only to get better. In the meantime one can try to correct faulty layout by tweaking baseline and/or kerning (when such option are available).

2.6 Pitfall: Blocks

The Unicode code space is subdivided into blocks of contiguous code points. For example, the block called Cyrillic runs from U+0400 till U+04FF. These blocks arose as an attempt at ordering the enormous amount of characters in Unicode, but the ideas of blocks very quickly ran into problems. First, the size of a block is fixed, so when a block is full, a new block will have to be instantiated somewhere further in the code space. For example, this led to the blocks Cyrillic Supplement, Cyrillic Extended-A (both of which are also already full) and Cyrillic

[7] For more details about OpenType, see http://www.adobe.com/products/type/opentype.html and http://www.microsoft.com/typography/otspec/. Additional systems for complex text layout are, among others, Microsoft's DirectWrite https://msdn.microsoft.com/library/dd368038.aspx and the open-source project HarfBuzz http://www.freedesktop.org/wiki/Software/HarfBuzz/.

[8] More information about AAT can be found at https://developer.apple.com/fonts/.
SIL's Graphite is described in detail at http://scripts.sil.org/cms/scripts/page.php?site_id=projects&item_id=graphite_home.

Extended-B. Second, when a specific character already exists, then it is not duplicated in another block, although the name of the block might indicate that a specific symbol should be available there. In general, names of blocks are just an approximate indication of the kind of characters that will be in the block.

The problem with blocks arises because finding the right character among the thousands of Unicode characters is not easy. Many software applications present blocks as a primary search mechanism, because the block names suggest where to look for a particular character. However, when a user searches for an IPA character in the block IPA Extensions, then many IPA characters will not be found there. For example, the velar nasal <ŋ> is not part of the block IPA Extensions because it was already included as latin small letter eng at U+014B in the block Latin Extensions-A.

In general, finding a specific character in the Unicode Standard is often not trivial. The names of the blocks can help, but they are not (and never were supposed to be) a foolproof structure. It is not the goal nor aim of the Unicode Consortium to provide a user interface to the Unicode Standard. If one often encounters the problem of needing to find a suitable character, there are various other useful services for end-users available.[9]

2.7 Pitfall: Names

The names of characters in the Unicode Standard are sometimes misnomers and should not be misinterpreted as definitions. For example, the combining grapheme joiner at U+034F does not join characters into larger graphemes (see Section 2.3) and the latin letter retroflex click U+01C3 is actually not the IPA symbol for a retroflex click, but for an alveolar click (see Section 3.4). In a sense, these names can be seen as "errors." However, it is probably better to realize that such names are just convenience labels that are not going to be changed. Just like the block names (Section 2.6), the character names are often helpful, but they are not supposed to be definitions.

The actual intended "meaning" of a Unicode codepoint is a combination of the name, the block and the character properties (see Section 1.4). Further details

[9] The Unicode website offers a basic interface to the code charts at http://www.unicode.org/charts/index.html. As a more flexible interface, we particularly like PopChar from Macility, available for both Macintosh and Windows. There are also various free websites that offer search interfaces to the Unicode code tables, like http://unicode-search.net or http://unicode-search.net. A further useful approach for searching characters using shape matching is http://shapecatcher.com.

about the underlying intentions with which a character should be used are only accessible by perusing the actual decisions of the Unicode Consortium. All proposals, discussions and decisions of the Unicode Consortium are publicly available. Unfortunately there is not (yet) any way to easily find everything that is ever proposed, discussed and decided in relation to a specific codepoint of interest, so many of the details are often somewhat hidden.[10]

2.8 Pitfall: Homoglyphs

Homoglyphs are visually indistinguishable glyphs (or highly similar glyphs) that have different code points in the Unicode Standard and thus different character semantics. As a principle, the Unicode Standard does not specify how a character appears visually on the page or the screen. So in most cases, a different appearance is caused by the specific design of a font, or by user-settings like size or boldface. Taking an example already discussed in Section 2.8, the following symbols <g *g* **g** g *g* **g**> are different glyphs of the same character, i.e. they may be rendered differently depending on the typography being used, but they all share the same code point (viz. latin small letter g at U+0067). In contrast, the symbols <AАΑᎪᗅᴀꓮ𐊠𝖠𝙰> are all different code points, although they look highly similar—in some cases even sharing exactly the same glyph in some fonts. All these different A-like characters include the following code points in the Unicode Standard:

<A> latin capital letter a, at U+0041
<А> cyrillic capital letter a, at U+0410
<Α> greek capital letter alpha, at U+0391
<Ꭺ> cherokee letter go, at U+13AA
<ᗅ> canadian syllabics carrier gho, at U+15C5
<ᴀ> latin small letter capital a, at U+1D00
<ꓮ> lisu letter a, at U+A4EE
<𐊠> carian letter a, at U+102A0
<𝖠> mathematical sans-serif capital a, U+1D5A0
<𝙰> mathematical monospace capital a, at U+1D670

[10] All proposals and other documents that are the basis of Unicode decisions are avaialbe at http://www.unicode.org/L2/all-docs.html. The actual decisions that make up the Unicode Standard are documented in the minutes of the Unicode Technical Committee, available at http://www.unicode.org/consortium/utc-minutes.html.

The existence of such homoglyphs is partly due to legacy compatibility, but for the most part these characters are simply different characters that happen to look similar.[11] Yet, they are supposed to behave different from the perspective of a font designer. For example, when designing a Cyrillic font, the <A> will have different aesthetics and different traditional expectation compared to a Latin <A>.

Such homoglyphs are a widespread problem for consistent encoding. Although for most users it looks like the words <voces> and <voces> are almost identical, in actual fact they do not even share a single code point.[12] For computers these two words are completely different entities. Commonly, when users with Cyrillic or Greek keyboards have to type some Latin-based orthography, they mix similar looking Cyrillic or Greek characters into their text, because those characters are so much easier to type. Similarly, when users want to enter an unusual symbol, they normally search by visual impression in their favorite software application, and just pick something that looks reasonably alike to what they expect the glyph to look like.

It is really easy to make errors at text entry and add characters that are not supposed to be included. Our proposals for orthography profiles (see Chapter 4) are a method for checking the consistency of any text. In situations in which interoperability is important, we consider it crucial to add such checks in any workflow.

2.9 Pitfall: Canonical equivalence

For some characters, there is more than one possible encoding in the Unicode Standard. This is a possible pitfall, as this would mean that for the computer there exist multiple different entities that for a user are the same. This would, for example, lead to problems with searching, as the computer would search for specific encodings, and not find all expected characters. As a solution, the Unicode Standard includes a notion of CANONICAL EQUIVALENCE. Different encodings are explicitly declared as equivalent in the Unicode Standard code tables. Further, to harmonize all encodings in a specific piece of text, the Unicode Standard proposes a mechanism of NORMALIZATION.

Consider for example the characters and following Unicode code points:

[11] A particularly nice interface to look for homoglyphs is http://shapecatcher.com, based on the principle of recognizing shapes (Belongie, Malik & Puzicha 2002).

[12] The first words consists completely of Latin characters, namely U+0076, U+006F, U+0063, U+0065 and U+0073, while the second is a mix of Cyrillic and Greek characters, namely U+03BD, U+03BF, U+0041, U+0435 and U+0455.

- <Å> LATIN CAPITAL LETTER A WITH RING ABOVE U+00C5
- <Å> ANGSTROM SIGN U+212B
- <Å> LATIN CAPITAL LETTER A U+0041 + COMBINING RING ABOVE U+030A

The character, represented here by glyph <Å>, is encoded in the Unicode Standard in the first two examples by a single-character sequence; each is assigned a different code point. In the third example, the glyph is encoded in a multiple-character sequence that is composed of two character code points. All three sequences are CANONICALLY EQUIVALENT, i.e. they are strings that represent the same abstract character and because they are not distinguishable by the user, the Unicode Standard requires them to be treated the same in regards to their behavior and appearance. Nevertheless, they are encoded differently. For example, if one were to search an electronic text (with software that does not apply Unicode Standard normalization) for ANGSTROM SIGN (U+212B), then the instances of LATIN CAPITAL LETTER A WITH RING ABOVE (U+00C5) would not be found.

In other words, there are equivalent sequences of Unicode characters that should be normalized, i.e. transformed into a unique Unicode-sanctioned representation of a character sequence called a NORMALIZATION FORM. Unicode provides a Unicode Normalization Algorithm, which essentially puts combining marks into a specific logical order and it defines decomposition and composition transformation rules to convert each string into one of four normalization forms. We will discuss here the two most relevant normalization forms: NFC and NFD.

The first of the three characters above is considered the NORMALIZATION FORM C (NFC), where C stands for composition. When the process of NFC normalization is applied to the character sequences in 2 and 3, both sequences are normalized into the PRE-COMPOSED character sequence in 1. Thus all three canonical character sequences are standardized into one composition form in NFC.The other central Unicode normalization form is the NORMALIZATION FORM D (NFD), where D stands for decomposition. When NFD is applied to the three examples above, all three, including importantly the single-character sequences in 1 and 2, are normalized into the DECOMPOSED multiple-sequence of characters in 3. Again, all three are then logically equivalent and therefore comparable and syntactically interoperable.

As illustrated, some characters in the Unicode Standard have alternative representations (in fact, many do), but the Unicode Normalization Algorithm can be used to transform certain sequences of characters into canonical forms to test for equivalency. To determine equivalence, each character in the Unicode Standard is associated with a combining class, which is formally defined as a character property called CANONICAL COMBINING CLASS which is specified in the Unicode

Character Database. The combining class assigned to each code point is a numeric value between 0 and 254 and is used by the Unicode Canonical Ordering Algorithm to determine which sequences of characters are canonically equivalent. Normalization forms, as very briefly described above, can be used to ensure character equivalence by ordering character sequences so that they can be faithfully compared.

It is very important to note that any software applications that is Unicode Standard compliant is free to change the character stream from one representation to another. This means that a software application may compose, decompose or reorder characters as its developers desire; as long as the resultant strings are canonically equivalent to the original. This might lead to unexpected behavior for users. Various players, like the Unicode Consortium, the W3C, or the TEI recommend NFC in most user-directed situations, and some software applications that we tested indeed seem to automatically convert strings into NFC.[13] This means in practice that if a user, for example, enters <a> and <◌̀>, i.e. LATIN SMALL LETTER A at U+0061 and COMBINING GRAVE ACCENT at U+0300, this might be automatically converted into <à>, i.e. LATIN SMALL LETTER A WITH GRAVE at U+00E0.[14]

2.10 Pitfall: Absence of canonical equivalence

Although in most cases canonical equivalence will take care of alternative encodings of the same character, there are a some cases in which the Unicode Standard decided against equivalence. This leads to identical characters that are not equivalent, like <ø> LATIN SMALL LETTER O WITH STROKE at U+00F8 and <ø> a combination of LATIN SMALL LETTER O at U+006F with COMBINING SHORT SOLIDUS OVERLAY at U+0037. The general rule followed is that extensions of Latin characters that are connected to the base character are not separated as combining diacritics. For example, characters like <ŋ ɲ ɳ> or <ɖ ɗ> are obviously derived from <n> and <d> respectively, but they are treated like new separate characters in the Unicode Standard. Likewise, characters like <ø> and <ƈ> are not separated into a base character <o> and <c> with an attached combining diacritic.

[13] See the summary of various recommendation here: http://www.win.tue.nl/~aeb/linux/uc/nfc_vs_nfd.html.

[14] The behavior of software applications can be quite erratic in this respect. For example, Apple's TextEdit does not do any conversion on text entry. However, when you copy and paste some text inside the same document in rich text mode (i.e. RTF-format), it will be transformed into NFC on paste. Saving a document does not do any conversion to the glyphs on screen, but it will save the characters in NFC.

Interestingly, and somewhat illogically, there are three elements, which are directly attached to their base characters, but which are still treated as separable in the Unicode Standard. Such characters are decomposed (in NFD normalization) in a base character with a combining diacritic. However, it is these cases that should be considered the exceptions to the rule. These three elements are the following:

- <◌̧>: the COMBINING CEDILLA at U+0327
  This diacritic is for example attested in the precomposed character <ç> LATIN SMALL LETTER C WITH CEDILLA at U+00E7. This <ç> will thus be decomposed in NFC normalization.
- <◌̨>: the COMBINING OGONEK at U+0328
  This diacritic is for example attested in precomposed <ą> LATIN SMALL LETTER A WITH OGONEK at U+0105. This <ą> will thus be decomposed in NFC normalization.
- <◌̛>: the COMBINING HORN at U+031B
  This diacritic is for example attested in precomposed <ơ> LATIN SMALL LETTER O WITH HORN at U+01A1. This <ơ> will thus be decomposed in NFC normalization.

There are further combinations that deserve specific care because it is actually possible to produce identical characters in different ways without them being canonically equivalent. In these situations, the general rule holds, namely that characters with attached extras are not decomposed. However, in the following cases the "extras" actually exist as combining diacritics, so there is also the possibility to construct a character by using a base character with those combining diacritics.

- First, there are the combining characters designated as "combining overlay" in the Unicode Standard, like <◌̴> COMBINING TILDE OVERLAY at U+0334 or <◌̵> COMBINING SHORT STROKE OVERLAY at U+0335. There are many characters that look like they are precomposed with such an overlay, for example <ł ƀ đ ꝑ> or <ɨ ɨ ɟ ɾ>, or also the example of <ø> given at the start of this section. However, they are not decomposed in NFD normalization.
- Second, the same situation also occurs with combining characters designated as "combining hook", like <◌̡> COMBINING PALATALIZED HOOK BELOW at U+0321. This element seems to occur in precomposed characters like <ҕ ɖ ʄ ҡ>. However, they are not decomposed in NFD normalization.

To harmonize the encoding in these cases it is not sufficient to use Unicode normalization. Additional checks are necessary, for example by using orthography profiles (see Chapter 4).

2.11 Pitfall: File formats

Unicode is a character encoding standard, but these characters of course actually appear inside some kind of computer file. The most basic Unicode-based file format is pure line-based text, i.e. strings of Unicode-encoded characters separated by line breaks (note that these line breaks are what for most people intuitively corresponds to paragraph breaks). Unfortunately, even within this apparently basic setting there exist a multitude of variants. In general, these different possibilities are well-understood in the software industry, and nowadays they normally do not lead to any problems for the end user. However, there are some situations in which a user is suddenly confronted with cryptic questions in the user interface involving abbreviations like LF, CR, BE, LE or BOM. Most prominently this occurs with exporting or importing data in several software applications from Microsoft. Basically, there are two different issues involved. First, the encoding of line breaks and, second, the encoding of the Unicode characters into code units and the related issue of endianness.

Line breaks

The issue with LINE BREAKS originated with the instructions necessary to direct a printing head of a physical printer to a new line. This involves two movements, known as CARRIAGE RETURN (CR, returning the printing head to the start of the line on the page) and LINE FEED (LF, moving the printing head to the next line on the page). Physically, these are two different events, but conceptually together they form one action. In the history of computing, various encodings of line breaks have been used (e.g. CR+LF, LF+CR, only LF, or only CR). Currently, all Unix and Unix-derived systems use only LF as code for a line break, while software from Microsoft still uses a combination of CR+LF. Today, most software applications recognize both options, and are able to deal with either encoding of line breaks (until rather recently this was not the case, and using the wrong line breaks would lead to unexpected errors). Our impression is that there is a strong tendency in software development to standardize on the simpler "only LF" encoding for line breaks, and we suggest that everybody use this encoding whenever possible.

Code units

The issue with CODE UNITS stems from the question how to separate a stream of binary ones and zero, i.e. bits, into chunks representing Unicode characters. A code unit is the sequence of bits used to encode a single character in an encoding. Depending on different use cases, the Unicode Standard offers three different approaches, called UTF-32, UTF-16 and UTF-8.[15] The details of this issue is extensively explained in section 2.5 of the Unicode Core Specification Unicode Consortium (2014).

Basically, UTF-32 encodes each character in 32 bits (32 *bi*nary uni*ts*, i.e. 32 zeros or ones) and is the most disk-space-consuming variant of the three. However, it is the most efficient encoding processing-wise, because the computer simply has to separate each character after 32 bits.

In contrast, UTF-16 uses only 16 bits per character, which is sufficient for the large majority of Unicode characters, but not for all of them. A special system of SURROGATES is defined within the Unicode Standard to deal with these additional characters. The effect is a more disk-space efficient encoding (approximately half the size), while adding a limited computational overhead to manage the surrogates.

Finally, UTF-8 is a more complex system that dynamically encodes each character with the minimally necessary number of bits, choosing either 8, 16 or 32 bits depending on the character. This represents again a strong reduction in space (particularly due to the high frequency of data using erstwhile ASCII characters, which need only 8 bits) at the expense of even more computation necessary to process such strings. However, because of the ever growing computational power of modern machines, the processing overhead is in most practical situations a non-issue, while saving on space is still useful, particularly for sending texts over the Internet. As an effect, UTF-8 has become the dominant encoding on the World Wide Web. We suggest that everybody uses UTF-8 as their default encoding.

A related problem is a general issue about how to store information in computer memory, which is known as ENDIANNESS. The details of this issue go beyond the scope of this book. It suffices to realize that there is a difference between BIG-ENDIAN (BE) storage and LITTLE-ENDIAN (LE) storage. The Unicode Standard offers a possibility to explicitly indicate what kind of storage is used by starting

[15] The letters UTF stand for UNICODE TRANSFORMATION FORMAT, but the notion of "transformation" is a legacy notion that does not have meaning anymore. Nevertheless, the designation UTF (in capitals) has become an official standard designation, but should probably best be read as simply "Unicode Format."

a file with a so-called BYTE ORDER MARK (BOM). However, the Unicode Standard does not require the usage of BOM, preferring other non-Unicode methods to signal to computers which kind of endianness is used. This issue only arises with UTF-32 and UTF-16 encodings. When using the preferred UTF-8, using a BOM is theoretically possible, but strongly dispreferred according to the Unicode Standard. We suggest that everyone tries to prevent the inclusion of BOM in your data.

2.12 Pitfall: Software

2.13 Recommendations

Summarizing the pitfalls discussed in this chapter, we propose the following recommendations:

- To prevent strange boxes instead of nice glyphs, always install a few fonts with a large glyph collection and at least one fall-back font (see Section 2.4).
- Unexpected visual impressions of symbols does not necessarily mean that the actual encoding is wrong. It is mostly a problem of faulty rendering (see Section 2.5).
- Do not trust the names of codepoints as a definition of the character (see Section 2.7). Also do not trust Unicode blocks as a strategy to find specific characters (see Section 2.6)
- To ensure consistent encoding of texts, apply Unicode normalization (NFC or NFD, see Section 2.9).
- To prevent remaining inconsistencies after normalization, for example stemming from homoglyphs (see Section 2.8) or from missing canonical equivalence in the Unicode Standard (see Section 2.10) use orthography profiles (see Chapter 4).
- To deal with "tailored" grapheme clusters (Section 2.3), use Unicode Locale Descriptions, or orthography profiles (see Chapter 4).
- As a preferred file format, use Unicode Format UTF-8 in Normalization Form Composition (NFC) with LF line endings, but without byte order mark (BOM), whenever possible (see Section 2.11). This last nicely cryptic recommendation has T-shirt potential:

I prefer it
UTF-8 NFC LF no BOM

3 IPA meets Unicode

3.1 Introduction

The International Phonetic Alphabet (IPA) is a common standard in linguistics to transcribe sounds of spoken language into discrete segments using a Latin-based alphabet. Although IPA is reasonably easily adhered to with pen and paper, it is not trivial to encode IPA characters electronically. Similar to the previous chapter, in this chapter we discuss various pitfalls with the encoding of IPA in the Unicode Standard. We will specifically refer to the 2005 version of the IPA (International Phonetic Association 2005) and the 7.0 version of Unicode (Unicode Consortium 2014).

The details of the encoding are unimportant as long as the transcription is only directed towards phonetically trained eyes. For a linguist reading an IPA transcription, many of the details that will be discussed in this chapter might seem like hair-splitting trivialities. However, if IPA transcriptions are intended to be used across resources (e.g. searching similar phenomena across different languages) then it becomes crucial that there are strict encoding guidelines. Our main goal in this chapter is to present the encoding issues and propose recommendations for a "strict" IPA encoding.

There are several pitfalls to be aware of when using the Unicode Standard to encode IPA. As we have said before, from a linguistic perspective it might sometimes look like the Unicode Consortium is making incomprehensible decisions, but it is important to realize that the consortium has tried and is continuing to try to be as consistent as possible across a wide range of use cases, and it does place linguistic traditions above other orthographic choices. Furthermore, when we look at the history of how the IPA met Unicode, we see that many of the decisions for IPA symbols in the Unicode Standard come directly from the International Phonetic Association itself. Therefore, many pitfalls that we will encounter have their grounding in the history of the principles of the IPA, as well as in the technological considerations involved in creating a single multilingual encoding. In general, we strongly suggest to linguists to not complain about any decisions in the Unicode Standard, but to try and understand the rationale

of the International Phonetic Association and the Unicode Consortium (both of which are almost always well-conceived in our experience) and devise ways to work with any unexpected behavior.

Many of the current problems derive from the fact that the IPA is clearly historically based on the Latin script, but different enough from most other Latin-based writing systems to warrant special attention. This ambivalent status of the IPA glyphs (partly Latin, partly special) is unfortunately also attested in the treatment of IPA in the Unicode Standard. In retrospect, it might have been better to consider the IPA (and other transcription systems) to be a special kind of script within the Unicode Standard, and treat the obvious similarity to Latin glyphs as a historical relic. All IPA glyphs would then have their own code points, instead of the current situation in which some IPA glyphs have special code points, while others are treated as being identical to the regular Latin characters. Yet, the current situation, however unfortunate, is unlikely to change, so as linguists we must learn to deal with the specific pitfalls of IPA within the Unicode Standard.

In the following sections, we will describe these pitfalls in detail. But first we present a brief history of IPA, its principles, its computational representation and the need for and challenge in creating a single multilingual computer encoding, and then how IPA meet Unicode.

3.2 The International Phonetic Alphabet (IPA)

3.2.1 Brief history

Established in 1886, the International Phonetic Association (henceforth Association) has long maintained a standard alphabet, the International Phonetic Alphabet (IPA), which is a common standard in linguistics to transcribe sounds of spoken languages. It was first published in 1888 as an international system of phonetic transcription for oral languages and for pedagogical purposes. The draft of 1887 contained phonetic values for English, French and German. Diacritics for length and nasalization were already present in this draft and the same symbols are still used today.

Originally, the IPA alphabet was a list of symbols with pronunciation examples from words in different languages. In 1900 the symbols were first organized into chart and were given phonetic feature labels for consonants (e.g. for manner of articulation: ‘plosives’, ‘nasales’, ‘fricatives’; for place of articulation: ‘bronchials’, ‘laryngeales’, ‘labiales’) and for vowels (e.g. ‘fermeées’, ‘mi-fermeées’, ‘mi-overtes’, ‘overtes’). Throughout the last centory, the structure of the chart has changed

with increases in phonetic knowledge, and thus, like notational systems in other scientific disciplines, the IPA reflects facts and theories (of phonetic knowledge) that have developed over time. It is natural then that the IPA is modified occasionally to accommodate scientific innovations and discoveries. (In fact, this is part of the Association's mandate.) These changes are captured in the revisions to the IPA.

Over the years there has been several revisions, many minor. For example, articulation labels – what often are call features even though the IPA avoids this term – have changed (e.g. terms like 'lips', 'throat' or 'rolled' are not used anymore). Phonetic symbol values have changed (e.g. voiceless is no longer marked by <h>). Symbols have been dropped (e.g. the caret diacritic denoting 'long and narrow' is no longer used). And many symbols have been added to reflect contrastive sounds found in the world's very diverse phonological systems.

Although IPA began its life as a pedagogical tool, from its earliest days the Association aimed to provide "a separate sign for each distinctive sound; that is, for each sound which, being used instead of another, in the same language, can change the meaning of a word" (The International Phonetic Association 1999: 27). Distinctive sounds became later known as PHONEMES and the IPA has developed historically into a notational devise with a strictly segmented phonemic view. A phoneme is an abstract theoretical notion derived from an acoustic signal as produced by speakers in the real world. Therefore the IPA contains a number of theoretical assumptions about speech and how to analyze speech in written form.

Phonetic analysis is based on two premises: that it is possible to describe the acoustic speech signal (i.e. sound waves) in terms of sequentially ordered discrete segments, and, that each segment can be characterized by an articulatory target.[1] Today, the IPA chart reflects a linguistic basis grounded in principles of phonological contrast. This fact is stated explicitly in several places, including in the *Report on the 1989 Kiel convention* published in the *Journal of the International Phonetic Association* (Association et al. 1989: 67-68):

> The IPA is intended to be a set of symbols for representing all the possible sounds of the world's languages. The representation of these sounds uses a set of phonetic categories which describe how each sound is made. These categories define a number of natural classes of sounds that operate

[1] A purely phonetic description is only derivable from instrumental data in high quality sound recordings. Once spoken language data are segmented, phonological consideration inextricably play a roll in transcription. In other words, phonetic observations beyond quantitative acoustic analysis are made in terms of some phonological framework.

> in phonological rules and historical sound changes. The symbols of the IPA are shorthand ways of indicating certain intersections of these categories.

and in the *Handbook of the International Phonetic Association* (**IPA1993**):

> … a symbol can be regarded as a shorthand equivalent to a phonetic description, and a way of representing the contrasting sounds that occur in a language. Thus [m] is equivalent to 'voiced bilabial nasal', and is also a way of representing one of the contrasting nasal sounds that occur in English and other languages. […] When a symbol is said to be suitable for the representation of sounds in two languages, it does not necessarily mean that the sounds in the two languages are identical.

Although the IPA provides symbols to unambiguously represent phonemes, it also aims to represent phonetic details. Since phonetic detail could potentially include things like 'deep voice', the IPA restricts phonetic detail to linguistically relevant aspects of speech. These principles and conventions for using the IPA are outlined in the Association's Handbook.

3.2.2 Principles

IPA transcription has essentially two parts. The first is a text containing IPA symbols and the second is a set of conventions (rules) for interpreting those symbols (and their combinations). The IPA is designed to meet practical linguistic needs and is used to transcribe the phonetic or phonological structure of languages. It is also used increasingly as a foreign language learning tool, as a standard pronunciation guide and as a tool for creating practical orthographies of previously unwritten languages.

The current construction and use of the IPA are guided by principles outlined in the *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet* (The International Phonetic Association 1999: 159), henceforth HB. The use of the symbols to represent a language's phonological system is guided by the principle of contrast; where two words are distinguishable by phonetic contrast, those contrasts should be transcribed with different symbols (graphemes not diacritics). Allophonic distinction falls under the rubric of diacritically-distinguished symbols, e.g. [stop̃] vs [sp^{h}ot]. In sum:

- different symbols (without diacritics) should be used whenever a language employs two contrastive sounds

- when two sounds in a language are not known to be contrastive, the same symbol should be used to represent these sounds; however, diacritics may be used to distinguish such sounds when necessary
- diacritics cannot be dispensed with entirely, so the Association recommends to limit their use to:
  - denoting length, stress and pitch
  - representing minute shades of sounds
  - obviating the design of a (large) number of new symbols when a single diacritic suffices (e.g. nasalized vowels, aspirated stops)

Thus, an IPA transcription *always* consists of "a set of symbols and a set of conventions for their interpretation".

In systematic transcription (as opposed to impressionistic), there is a division between phonemic and allophonic transcription. The terms phoneme and allophone contain theoretical baggage, but the basic goal of a phonemic transcription is to distinguish all words in a language with the minimal number of transcription symbols (**Abercrombie1964**). Allophonic transcription uses a broader set of distinct symbols to describe systematic allophonic differences in sounds in words. These two systematic transcriptions are related to each other by a set of conventions in the IPA tradition, so that they can be converted between one and another.

An IPA transcription is connected to a speech event by a set of conventions. A phonetic (or impressionistic) transcription may use the conventions implicit in the IPA chart, i.e. the transcriber can indicate that the phonetic value of <ŋ͡m> is a simultaneous labial and velar closure which is voiced and contains nasal airflow.

A phonemic transcription includes the conventions of a particular language's phonological rules. These rules determine the realization of that language's phonemes. However, there can be different systems of phonemic transcription for the same variety of a language. The differences may result from the fact that more than one phonetic symbol may be appropriate for a phoneme (see Section **??**). Or the differences may be due to different phonemic analyses, e.g. Standard German's vowel system is arguably contrastive in length or tenseness.

An important principle of the IPA is that different representations resulting from different symbols and different analyses are in line with the IPA's aims. In other words, the IPA does not provide phonological analyses for specific languages and the IPA does not define a single "correct" transcription system. Rather, the IPA aims to provide a resource that allows users to express any analysis so that it is widely understood. Thus the IPA suits many linguists' needs because:

- it is intended to be a set of symbols for representing all possible sounds in the world's (spoken) languages
- its chart has a linguistic basis (specifically a phonological bias) rather than a general phonetic notation scheme
- its symbols can be used to represent distinctive feature combinations[2]
- its chart provides a summary of linguists' agreed-upon phonetic knowledge (a common denominator of phonological "facts")

Several styles of transcription with IPA are possible and the HB illustrates these and notes that they are all valid (see the 29 languages and their transcriptions in the original and initial *Illustrations of the IPA* (The International Phonetic Association 2007: 41–154)). Therefore, there are different but equivalent transcriptions, or as Ladefoged 1990: 64 captures it, "Perhaps now that the Association has been explicit in its eclectic approach, outsiders to the Association will no longer speak of *the* IPA transcription of a given phenomenon, as if there were only one approved style.".

Clearly not all phoneticians agree (or will likely ever agree of course) on all aspects of the IPA or on transcription practices. As noted above, there have been several revisions in the IPA's long history, but the current version (2005) is strikingly similar to the 1926 version. In 1989 an IPA revision convention was held in Kiel, Germany. As per other revisions, there was expansion and revisions to phonetic symbols in the IPA chart. Notably the marking of tone was extended with the addition of a second system for marking linguistic tones (Chao tones). Importantly, however, this was the first revision to address issues of computational representations for the IPA – the principles above of which have had several ramifications that make the interoperability of electronic linguistic data extremely difficult.

3.2.3 Computational representation

Prior to the Kiel Convention for the modern revision of the IPA in 1989, Wells 1987 collected and published practical approaches to coded representations of the IPA, which dealt mainly with the assignment of characters on the keyboard. The process of assigning standardized "computer codes" to phonetic symbols was assigned to the Workgroup on Computer Coding (henceforth working group) at

[2] Although the chart uses traditional manner and place of articulation labels, the symbols can nevertheless represent any defined bundle of features, binary or otherwise, to define phonetic dimensions.

the Kiel Convention. This working group was tasked with (Esling 1990; Esling & Gaylord 1993):

- determining how to represent the IPA numerically
- developing a set of numbers to refer to the IPA symbols unambiguously
- providing each symbol a unique name (intended to provide a mnemonic description of that character's shape)

The identification of IPA symbols with unique identifiers was a first step in formalizing the IPA computationally because it would give each symbol an unambiguous numerical identifier called an IPA Number. The numbering system was to be comprehensive enough to support future revisions of the IPA, including symbol specifications and diacritic placement. The application of diacritics was also to be made explicit.

Although the Association had never officially approved a set of names for the IPA symbols, each IPA symbol received a unique IPA Name. Many symbols already had an informal name (or two) used by linguists, but consensus on symbol names was growing due to the recent publication of the *Phonetic Symbol Guide* (Pullum & Ladusaw 1986). Thus most of the IPA symbol names were taken from Pullum & Ladusaw 1986 (The International Phonetic Association 1999: 31).

The working group insightfully decided that the computing-coding convention for the IPA should be independent of computer environments or formats (e.g. ASCII), i.e. the IPA Number was not meant to be implemented directly in a computer encoding. The working group report's declaration includes the explanatory remarks (Association et al. 1989: 82):

> The recommendation of a 7-bit ASCII or 8-bit extended-ASCII coding system would be short-sighted in view of development towards 16-bit and 32-bit processors. In fact, any specific recommendations would tie the Association to a stage of technological development which is bound to be outdated long before the next revision of the handbook.

Thus the coding convention was not meant to address the engineering aspects of the actual encoding in computers (cf. Anderson 1984). However, it was meant to serve as a basis for a communication-interchange standard for creating mapping tables from various computer encodings, fonts, phonetic-character-set software, etc., to common IPA Numbers, and thus symbols.[3]

[3] Remember, at this time in the late 1980s there was no stable multilingual computing environment. But some solution was needed because scholars were increasingly using personal

Furthermore, the assignment of computer codes to IPA symbols was meant to represent an unbiased formulation. The Association plays the role of an international advisory body and it stated that it should not recommend a particular existing system of encoding. In fact, during this time there were a number of coding systems used, but none of them had a dominant international position. The differences between systems were also either too great or too subtle to warrant an attempt at combining them (Association et al. 1989).

The working group assigned each IPA symbol to a unique three-digit number, i.e. an IPA Number. Encoded in this number scheme is information about the status of each symbol (see below). The IPA numbers are listed with the IPA symbols and they are also illustrated in IPA chart form (see Esling & Gaylord 1993: 84 or The International Phonetic Association 2007: App. 2). The numbers were assigned in linear order (e.g. [p] 101, [b] 102, [t] 103…) following the IPA revision of 1989 and its update in 1996.

The working group made the decision that no IPA symbol, past or present, could be ignored. The comprehensive inclusion of all IPA symbols was to anticipate the possibility that some symbols might be added, withdrawn, or reintroduced into current or future usage. For example, in the 1989 revision voiceless implosives < ɓ, ƭ, ƈ, ƙ, ʠ > were added; in the 1993 revision they were removed. Ligatures like < ʧ, ʤ > are included as formerly recognized IPA symbols; they are assigned to the "200 series" of IPA numbers as members of the group of symbols formerly recognized by the IPA. To ensure backwards compatibility, legacy IPA symbols would retain an IPA Number and an IPA Name for reference purposes. As we discuss below, this decision is later reflected in the Unicode Standard; many legacy IPA symbols reside in the IPA Extensions block.

The IPA Number is simply expressed as a "three-digit number numerical directory of digit triples" [4] The numbering scheme specifies three-digit codes, the first digit of which indicates the symbol's category Esling 1990; Esling & Gaylord 1993:

- 100s for accepted IPA consonants
- 200s for former IPA consonants and non-IPA symbols
- 300s for vowels
- 400s for segmental diacritics

computers for their research and many were quickly adopting electronic mail or discussion boards like Usenet as a medium for international exchanges. This was before the Internet as we know it today. Most of these systems ran on 8-bit hardware systems using a 7-bit ASCII character encoding.

[4] For practical purposes, the IPA Number also served as a typesetter's guide to the IPA chart.

- 500s for suprasegmental symbols
- 600s-800s for future specifications
- 900s for escape sequences

After a symbol is categorized, it is assigned a number sequentially, e.g. [i] 301, [e] 302, [ɛ] 303. The system allows for the addition of new symbols within the various series by appending them, e.g. [☒] 184. Former or often used but non-IPA symbols for consonants, vowels and diacritics are numbered from x99 backwards. For example, the voiceless and voiced postalveolar affricates and fricatives < č, ǰ, š, ž > are assigned the IPA numbers 299, 298, 297 and 296, respectively, because they are not sanctioned IPA symbols.

The assignment of the IPA numbers to IPA symbols provided the basis for uniquely identifying the set of past and present IPA symbols as a type of computational representational standard of the IPA. Within each revision of the IPA, the coding defines a closed and clearly defined set of characters. The benefits of this standardization are clear in at least two ways: it is used in translation tables that reference ASCII representations of the IPA, and this early computational representation of the IPA became the basis for its inclusion into the Unicode Standard version 1.0.

3.2.4 SAMPA and X-SAMPA

True to the working group's aim, the IPA numbers provided a mechanism for a communication interchange standard for creating mapping tables to various computer encodings. For example, the IPA coding system was used as a mapping system in the creation of SAMPA (Wells et al. 1992), an ASCII representation of the IPA symbols.

For a long time, linguists, like all other computer users, were limited to ASCII-encoded 7-bit characters, which only includes Latin characters, numbers and some punctuation and symbols. Restricted to these standard character sets that lacked IPA support or other language-specific graphemes that they needed, linguists devised their own solutions.[5] For example, some chose to represent unavailable graphemes with substitutes, e.g. the combination of <ng> to represent <ŋ>. Tech-savvy linguists redefined selected characters from a character encod-

[5] Early work addressing the need for a universal computing environment for writing systems and their computational complexity is discussed in Simons (1989). A survey of practical recommendations for language resources, including notes on encoding, can be found in Bird & Simons (2003)

ing by mapping custom made fonts to specific code points.[6]. However, one linguist's electronic text would not render properly on another linguist's computer without access to the same font. Furthermore, if two character encodings defined two character sets differently, then data could not be reliably and correctly displayed. This is a commonly encountered example of the non-interoperability of data and data formats.

One solution was the ASCII-ification of the IPA, which simply involved defining keyboard-able sequences as IPA symbol codings.[7] A successful effort was SAMPA (Speech Assessment Methods Phonetic Alphabet), which was created between 1988–1991 in Europe to represent IPA symbols with ASCII character sequences (Wells 1987; Wells et al. 1992), e.g. <p\> for [ɸ]. SAMPA was developed by a group of speech scientists from nine countries in Europe and it constituted the ASCII-IPA symbols needed for phonemic transcription of the principal European Union languages (Wells 1995). It is still widely used in language technology.

Two problems with SAMPA are that (i) it is only a partial encoding of the IPA and (ii) it encodes different languages in separate data tables, instead of using a universal alphabet, like IPA. SAMPA tables were developed as part of a European Commission-funded project to address technical problems like electronic mail exchange (what is now simply called email). SAMPA is essentially a hack to work around displaying IPA characters, but it provided speech technology and other fields a basis that has been widely adopted and often still used in code. So, SAMPA is a collection of tables to be compared, instead of a large universal table representing all languages.

An extended version of SAMPA, called X-SAMPA, set out to include every symbol, including all diacritics, in the IPA chart (Wells 1995). X-SAMPA is considered more universally applicable because it consists of one table that encodes all characters in IPA. In line with the principles of the IPA, SAMPA and X-SAMPA include a repertoire of symbols. These symbols are intended to represent phonemes rather than all allophonic distinctions. Additionally, both ASCII-ifications of IPA are useful because strings of SAMPA or X-SAMPA are (reportedly) uniquely parsable (Wells 1995). However, like the IPA, X-SAMPA has different notations for encoding the same phonetic phenomena (see Pitfall **??**).

SAMPA and X-SAMPA have been widely used for speech technology and as an encoding system in computational linguistics. In fact, they are stilled used

[6] For example, SIL's popular font SIL IPA 1990

[7] Wells 1987 provides an in-depth description of IPA codings from country-to-country. Later ASCII-IPAs include Kirshenbaum (created in 1992 in a Usenet group and named after its lead developer who was at Hewlett-Packard Laboratories) and Worldbet (published in 1993 by Hieronymus 1993, who was at AT&T Laboratories).

in popular software packages that require ASCII input and some of which have been co-opted for linguistic analyses, e.g. RuG/L04 and SplitsTree4.[8]

3.2.5 The need for a single multilingual environment

During the 1980's, it became increasingly clear that an adequate solution to the problem of multilingual computing environments was needed. Linguists were on the forefront of addressing this issue because they faced these challenges head-on by wishing to publish and communicate electronic text with phonetic symbols which were not included in basic ASCII.[9]

Long familiar were linguists already with the distinction between function and form. Even in the context of the computer implementation of writing systems, the necessity to distinguish form and function had been made (Becker 1984). The computer industry, on the other hand, did not consider, ignored, or simply did not encode this principle when creating operating systems like MS-DOS, which were limited to 256 code points (due to computer hardware architecture) and encoded with one-to-one mappings from character codes to graphemes.

Industry was starting to tackle the issues involved in developing a single multilingual computing environment on a variety of fronts, including the then new technology of bitmap fonts and the creation of Font Manager and Script Manager by Apple (Computer 1985, 1986, 1988). As noted above, around this time linguists were developing work-arounds such as SAMPA, so that they could communicate IPA transcription and use ASCII-based software. Some linguists formalized the issues of multilingual text processing from a computational perspective (Anderson 1984; Becker 1984; Simons 1989). The study of writing systems was also being invigorated by the computational challenges in making computers work in a multilingual environment.[10]

The second major benefit of the standardization of the IPA in a computational representation by the Kiel working group is that it provided the basis for a formal proposal to be submitted to various international standards organizations, several of which were trying to tackle (and in a sense 'win') the multilingual computing environment problem. Basically, everyone from corporations to governments to language scientists (for lack of a better term) wanted a single unified multilingual character encoding set for all the world's writing systems, even if

[8] See http://www.let.rug.nl/kleiweg/L04/ and http://www.splitstree.org/, respectively

[9] One only needs to look at facsimiles of older electronic documents to see exotic symbols written in by hand.

[10] **Sampson1985** urges linguists to view the study of writing systems as legitimate scientific enquiry.

they did not understand or appreciate the challenges involved in creating and adopting a solution. Additionally, advancements in computer hardware were making the solutions easier to implement in software.

The engineering problems and solutions had been spelled out years before, e.g. a two-byte encoding for multilingual text (Anderson 1984). Although languages vary to an astounding extent (cf. Evans & Levinson 2009), writing systems are quite similar formally and the issue of formal representation of the world's orthographic systems had also been addressed (Simons 1989).

A major obstacle in creating a single encoding multilingual environment from the perspective of writing systems involves the distinction between function and form (Becker 1984) This distinction is so central to basic linguistic theory and that trained linguists and semiologists take it as second nature. A central challenge in developing a universal character set was to combine a technological solution with a formalization of writing systems proper.[11]

In hindsight it is easy to lose sight of how impactful 30 years of technological development have been on linguistics, from theory development using quantitative means to pure data collection and dissemination (which as fed back into the former). But at the end of the 1970s, virtually no ordinary working linguist was using a person computer Simons 1996. Personal computer usage, however, dramatically increased throughout the 1980s. By 1990, dozens of character sets were in common use. They varied in their architecture and in their character repertoires, which made things a mess. There were two major players in the universal character set race: Unicode and the International Organization for Standardization (ISO).

3.2.6 Unicode and ISO 10646

In the late 1980s, a universal character set was being developed by what is now referred to as the Unicode Consortium.[12] This consortium consisted largely, although not entirely, of major US corporations, with the aim of overcoming the inoperability of different coded character sets and their costly hinderance for developing multilingual software development and for internationalization efforts. Commercial importance of course drove the early inclusion of Latin, non-Latin, and some exotic scripts; see the table of commercial importance as measured by GDP of countries using certain writing systems (**unicode88**).

[11] Of course there were additional practical issues to overcome, e.g. funding, creating the formal and technological proposal, deciding which characters and writing systems to include initially, while setting precedence of how to add new ones in the future.

[12] The Unicode Consortium was officially incorporated in January 1991.

The original Unicode manifesto is **unicode88.pdf**[13] Its aim was for a reliable international multilingual text encoding standard that would encompass all scripts of the world, or in the author's own words, "a new, world-wide ASCII". An in-depth history of Unicode, highlighting interesting facts like its first text prototypes at Apple and its incorporation into TrueType, is retold online.[14]

Unicode 88 provided the basic principles for the Unicode Standard's design – pushing for 16 bit representations of characters with a clear distinction between characters and glyphs. Some of the contents of this status proposal of 1988 were reworked for inclusion in the early Unicode Standard pre-publication drafts and by August 1990, the proposal was in a (very) rough draft format. Its editors and the Unicode Working Group (the predecessor to the Unicode Technical Committee) worked together to lay out the the proposed standard's structure and content. At this time, the proposal contained no code charts nor block descriptions.

The other major player in developing a universal character set was the ISO working group from the International Standards Organization (ISO), based in Europe, which was responsible for ISO/IEC 10646. This character set standard was composed in 1989 and a draft was published in 1990.[15] The 'Universal Multiple-Octet Coded Character Set' or simply UCS was the first officially standardized character encoding with the aim of including all characters from all writing systems.[16]

ISO/IEC 10646 is partly based on ISO/IEC 8859, a series of of ASCII-based standard character encodings published in 1987 that use a single bit 8-byte character set. Each part of the standard, e.g. 8859-1, 8859-5, 8859-6, encodes characters to support different languages' writing systems, e.g. Latin-1 Western European, Latin/Cyrillic, Latin/Arabic, respectively. Being a joint effort by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC), the aim of the standard is reliable information exchange. So, again, issues of phonetic symbol encoding, typography, etc., were ignored – or perhaps more properly put, not commercially driven at this early stage.

Intended for the major Western European languages, ISO/IEC 8859 was an extension of the ASCII character encoding standard, which included the English alphabet, numerals and computer control characters (e.g. beep, space, carriage return). By extending ASCII's 7-bit system to 8-bit, the character repertoire of each of ISO/IEC 8859 character set was doubled from 128 to 256 characters. Each char-

[13] http://www.unicode.org/history/unicode88.pdf
[14] http://www.unicode.org/history/earlyyears.html
[15] http://www.iso.org/iso/catalogue_detail.htm?csnumber=56921
[16] http://www.nada.kth.se/i18n/ucs/unicode-iso10646-oview.html

acter set defined a mapping between digital bit patterns and characters, which are visually rendered on screen as graphic symbols. ASCII was shared between ISO/IEC 8859 character sets, but the characters in the extra bit patterns were different. Thus an aim of the ISO working group responsible for ISO/IEC 10646 was to bring all characters in all writings systems into a single unified encoding.

In 1991, the Unicode Consortium and the ISO Working Group for ISO/IEC 10646 decided to create a single universal standard for encoding multilingual text.[17] The two character sets converged, resulting in mutually acceptable changes to both, and each group keeps versions of their respective character codes and encoding forms synchronized.[18] Although each standard has its own form of reference and the terminology in each may differ slightly, the practical difference is that the Unicode Standard is a formal implementation of ISO/IEC 10646 and imposes additional constraints on its implementation. The Unicode Standard includes character data, algorithms and specifications, outside the scope of ISO/IEC 10646, which ensure, when properly implemented in software applications and platforms, that characters are treated uniformly.

The incorporation of the Unicode Standard into the international encoding standard ISO 10646 was approved by ISO as an International Standard in June 1992.[19] The joint Unicode and ISO/IEC 10646 standard has become *the* universal character set and it is a single multilingual environment for the majority of the world's written languages. Its formal implementation has also been vital to the rise of a multi-lingual Internet.

3.2.7 IPA and Unicode

It was a long journey, but the goal of achieving a single multilingual computing environment has largely been accomplished. We users, however, must cope with the pitfalls that were dug along the way (see Pitfall sections below). Some linguists, including your humble authors, are particularly sensitive to these issues. So we provide practical advice and approaches in the rest of this chapter. But first, we explain how the IPA became incorporated into the Unicode Standard via ISO/IEC 10646.

After the Kiel Convention in 1989, the Computer Coding of the IPA working group assisted the International Phonetic Association in representing the IPA to ISO and to the Text Encoding Initiative (TEI) (Esling & Gaylord 1993). The working group's formalization of the IPA, i.e. a full listing of agreed upon computer

[17] http://unicode.org/book/appC.pdf
[18] http://www.unicode.org/versions/
[19] http://www.unicode.org/versions/Unicode1.0.0/Notice.pdf

codings for phonetic symbols, was used in developing writing systems descriptions which were at the time being solicited for scripts to be included in initiatives for new multilingual international character encoding standards. The working group for ISO/IEC 10646 and Unicode were two such initiatives.

In the historical context of the IPA being considered for inclusion in ISO/IEC 10646, it is important to realize that there were a variety of sources (i.e. not just from the Association) which submitted character proposals for phonetic alphabets. These proposals, including from the Association via the Kiel working group, were considered as a whole by the ISO working groups which were responsible for incorporating a phonetic script into the universal character set (UCS). The ISO working groups that were responsible for assigning a phonetic character set then made their own submissions as part of a review process by ISO for approval based on both "informatic" and phonetic criteria (Esling & Gaylord 1993: 86).

Character set ISO/IEC 10646 was approved by ISO, including the phonetic characters submitted to them in May 1993. The set of IPA characters were assigned UCS codes in 16 bit representation (in hexadecimal) and were published Tables 2 and 3 in Esling & Gaylord 1993, which include a graphical representation of the IPA symbol, its IPA Name, phonetic description, IPA Number, UCS Code and AFII Code.[20] Because the character sets of ISO/IEC 10646 and the Unicode Standard converged, the IPA as submitted by the Association and reviewed and further submitted by the ISO working group, was included in the Unicode Standard Version 1.0 – largely as we know it today.[21]

With subsequent revisions to the IPA, one might expect the Unicode Consortium would update the Unicode Standard in a way that is inline with linguists' or other language scientists' intuitions. However, updates that go against the ISO's and the Unicode Standard's principles of maintaining backwards compatibility lose out, i.e. it is more important to deal with the pitfalls created along the way then it is to change the standard. Therefore, many of the pitfalls we encounter when using Unicode IPA are historic relics that we have to come to grips with.

[20] The Association for Font Information Interchange (AFII) was an international database of glyphs created to promote the standardization of font data required to produce ISO/IEC 10646.

[21] The Association later made the foresightful remark, "When this character set is in wide use, it will be the normal way to encode IPA symbols." (The International Phonetic Association 1999: 164).

3.3 Pitfall: No complete IPA code block

The ambivalent nature of IPA glyphs arises because, on the one hand, the IPA uses Latin-based glyphs like <a>, <b> or <p>. From this perspective, the IPA seems to be just another orthographic tradition using Latin characters, all of which do not get a special treatment within the Unicode Standard (just like e.g. the French, German, or Danish orthographic traditions do not have a special status). On the other hand, the IPA uses many special symbols (like turned <ɐ>, mirrored <ɘ> and/or extended <ɦ> Latin glyphs) not found in any other Latin-based writing system. For this reason a special block with code points, called IPA Extensions was already included in the first version of the Unicode Standard (Version 1.0 from 1991).

As explained in Section 2.6, the Unicode Standard code space is subdivided into character blocks, which generally encode characters from a single script. However, as is illustrated by the IPA, characters that form a single writing system may be dispersed across several different character blocks. With its diverse collection of symbols from various scripts and diacritics, the IPA is spread across 12 blocks in the Unicode Standard:[22]

- Basic Latin (27 characters)
  a b c d e f h i j k l m n o p q r s t u v w x y z . |
- Latin-1 Supplement (4 characters)
  æ ç ð ø
- Latin Extended-A (3 characters)
  ħ ŋ œ
- Latin Extended-B (4 characters)
  ǀ ǁ ǂ ǃ
- Latin Extended-C (1 character):
  ⱱ
- IPA Extensions (67 characters)
  ɐ ɑ ɒ ɓ ɔ ɕ ɖ ɗ ɘ ə ɛ ɜ ɞ ɟ ɠ ɡ ɢ ɣ ɤ ɥ ɦ ɧ ɨ ɪ ɬ ɭ ɮ ɯ ɰ ɱ ɲ ɳ ɴ
  ɵ ɶ ɸ ɹ ɻ ɺ ɽ ɾ ʁ ʂ ʃ ʄ ʈ ʉ ʊ ʋ ʌ ʍ ʎ ʏ ʐ ʑ ʒ ʔ ʕ ʘ ʙ ʗ ʜ ʝ ʟ ʡ ʢ

[22] This number of blocks depends on whether only IPA-sanctioned symbols are counted or if the phonetic symbols commonly found in the literature are also included, see Moran 2012: Appendix C. The 159 characters from 12 code blocks shown here are the characters proposed for "strict" IPA encoding, as discussed in Section 3.14.

- Greek and Coptic (3 characters)
  β θ χ
- Spacing Modifier Letters (17 characters)
  ◌ˠ ◌ˡ ◌ʷ ◌ʲ ◌ˠ ◌ˤ ◌ʰ ◌’ ◌ː ◌ˑ ◌˥ ◌˦ ◌˧ ◌˨ ◌˩ ◌ˈ ◌ˌ
- Superscripts and Subscripts (1 character)
  ◌ⁿ
- Combining Diacritical Marks (25 characters)
  ◌̊ ◌̤ ◌̰ ◌̪ ◌̺ ◌̻ ◌̹ ◌̜ ◌̟ ◌̠ ◌̈ ◌̽ ◌̩ ◌̯ ◌̫ ◌̬ ◌̝ ◌̞ ◌̘ ◌̙ ◌̃ ◌̋ ◌̆ ◌̌ ◌̍ ◌͡◌
- General Punctuation (2 characters)
  ‖ ‿
- Arrows (4 characters)
  ↑ ↓ ↗ ↘

3.4 Pitfall: IPA homoglyphs in Unicode

Another problem is the large number of homoglyphs, i.e. different characters that have highly similar glyphs (or even completely identical glyphs, depending on the font rendering). For example, a user of a Cyrillic computer keyboard should ideally not use the <а> cyrillic small letter a at code point U+0430 for IPA transcriptions, but instead use the <a> latin small letter a at code point U+0061, although visually they are mostly indistinguishable, and the Cyrillic character is more easily typed on a Cyrillic keyboard. Some further problematic homoglyphs related to encoding IPA in the Unicode Standard are the following.

- The uses of the apostrophe has led to long discussions on the Unicode Standard email list. An English keyboard inputs the symbol <◌'> apostrophe at U+0027, although the preferred Unicode apostrophe is the <◌’> right single quotation mark at U+2019.[23] However, the glottal stop/-glottalization/ejective marker is yet another completely different character, namely <◌’>, i.e. the modifier letter apostrophe at U+02BC, which unfortunately looks mostly extremely similar to U+2019.
- Another problem is the <◌ˁ> modifier letter reversed glottal stop at U+02C1 vs. the <◌ˤ> modifier letter small reversed glottal stop

[23] Note that many word processors (like Microsoft Word) by default will replace straight quotes by curly quotes, depending on the whitespace around it.

at U+02E4. Both appear in various resources representing phonetic data online. This is thus a clear example for which the Unicode Standard does not solve the linguistic standardization problem.

- Linguists are also unlikely to distinguish between the <ə> LATIN SMALL LETTER SCHWA at code point U+0259 and <ɘ> LATIN SMALL LETTER TURNED E at U+01DD.
- The alveolar click <!> at U+01C3 is of course often simply typed as <!> EXCLAMATION MARK at U+0021.[24]
- The dental click <ǀ>, in Unicode known as LATIN LETTER DENTAL CLICK at U+01C0, is often simply typed as <|> VERTICAL LINE at U+007C.
- For the marking of length there is a special Unicode characters, namely <◌ː> MODIFIER LETTER TRIANGULAR COLON at U+02D0. However, typing <◌:> COLON at U+003A is of course much easier.

Conversely, non-linguists are unlikely to distinguish any semantic difference between an open back unrounded vowel <ɑ> LATIN SMALL LETTER ALPHA at U+0251, and the open front unrounded vowel <a> LATIN SMALL LETTER A at U+0061, basically treating them as homoglyphs, although they are different phonetic symbols. But even among linguists this distinction leads to problems. For example, as pointed out by Mielke (2009), there is a problem stemming from the fact that about 75% of languages are reported to have a five-vowel system (Maddieson 1984). Historically, linguistic descriptions tend not to include precise audio recording and measurements of formants, so this may lead one to ask if the many <a> characters that are used in phonological description reflects a transcriptional bias. The common use of <a> in transcriptions could be in part due to the ease of typing the letter on an English keyboard (or for older descriptions, the typewriter). We found it to be exceedingly rare that a linguist uses <ɑ> for a low back unrounded vowel.[25] They simply use <a> as long as there is no opposition to <ɑ>.

Making things even more problematic, there is an old typographic tradition that the double-story <a> uses a single-story <ɑ> in italics. This leads to the unfortunate effect that even in many well-designed fonts the italics of <a> and

[24] In the Unicode Standard the <!> at U+01C3 is labeled LATIN LETTER RETROFLEX CLICK, but in IPA that glyph is used for an alveolar or postalveolar click (not retroflex). This naming is probably best seen as an error in the Unicode Standard. For the "real" retroflex click, see Section 3.13.

[25] One example is **Vidal2001a** in which the author states: "The definition of Pilagá /a/ as [+back] results from its behavior in certain phonological contexts. For instance, uvular and pharyngeal consonants only occur around /a/ and /o/. Hence, the characterization of /a/ and /o/ as a natural class of (i.e., [+back] vowels), as opposed to /i/ and /e/."

<ɑ> use the same glyph. For example, in Linux Libertine (the font of this book) the italics of these characters are highly similar <*a*> and <*ɑ*>, while in Charis SIL they are identical: <*a*> and <*ɑ*>. If this distinction has to be kept upright in italics, the only solution we can currently offer is to use SLANTED glyphs (i.e. artificially italicized glyphs) instead of real italics (i.e. special italics glyphs designed by a typographer).[26]

3.5 Pitfall: Homoglyphs in IPA

Reversely, there are a few cases in which the IPA distinguishes different phonetic concepts, but the visual characters used by the IPA look very much alike. Such cases are thus homoglyphs in the IPA itself, which of course need different encodings.

- The dental click <|> and the indication of a minor group break <|> look almost the same in most fonts. For a proper encoding, the LATIN LETTER DENTAL CLICK at U+01C0 and the VERTICAL LINE at U+007C should be used, respectively.
- Similarly, the alveolar lateral click <‖> should be encoded with a LATIN LETTER LATERAL CLICK at U+01C1, different from <‖>, which according to the IPA is the character to by used for a major group break (by intonation), to be encoded by DOUBLE VERTICAL LINE at U+2016.
- The marking of primary stress < ˈ > looks like an apostrophe, and is often typed with the same symbol as the ejective <◌’>. For a proper encoding, these two symbols should be typed as MODIFIER LETTER VERTICAL LINE at U+02C8 and MODIFIER LETTER APOSTROPHE at U+02BC, respectively.

[26] For example, the widely used IPA font Doulos SIL (http://scripts.sil.org/cms/scripts/page.php?item_id=DoulosSIL) does not have real italics. This leads some word-processing software, like Microsoft Word, to produce slanted glyphs instead. That particular combination of font and software application will thus lead to the desired effect distinguishing <a> from <ɑ> in italics. However, note that when the text is transferred to another font (i.e. one that includes real italics) and/or to another software application (like Apple Pages, which does not perform slanting), then this visual appearance will be lost. In this case we are thus still in the pre-Unicode situation in which the choice of font and rendering software actually matters. The ideal solution from a linguistic point of view would be the introduction of a new IPA code point for a different kind of <a>, which explicitly specifies that it should still be rendered as a double-story character when italicized. After informal discussion with various Unicode players, our impression is that this highly restricted problem is not sufficiently urgent to introduce even more <a> homographs in Unicode (which already lead to much confusion, see Section 2.8).

- There are two different "dashed"-l characters in IPA, namely the <ɬ> LATIN SMALL LETTER L WITH MIDDLE TILDE at U+026B and the <ɬ> LATIN SMALL LETTER L WITH BELT at U+026C. These of course look highly similar, although they are different sounds. As a solution, we will actually propose to not use the middle tilde at all (see Section 3.6).

3.6 Pitfall: Multiple encoding options in IPA

It is not just the Unicode Standard that offers multiple options for encoding the IPA. Even the IPA specification itself offers some flexibility in how transcriptions have to be encoded. There are a few cases in which the IPA explicitly allows for different options of transcribing the same phonetic content. This is understandable from a transcriber's point of view, but it is not acceptable when the goal is interoperability between resources written in IPA. We consider it crucial to distinguish between "lax" IPA, for which it is sufficient that any phonetically-trained reader is able to understand the transcription, and "strict" IPA, which should be standardized on a single unique encoding for each sound, so search will work across resources. We are aware of the following non-unique encoding options in the IPA, which will be discussed in turn below:

- The marking of tone
- The marking of <g>
- The marking of velarization and pharyngealization
- The placement of diacritics

The first case in which the IPA allows for different encodings is the question of how to transcribe tone. There is an old tradition to use diacritics on vowels to mark different tone levels, e.g. <ȅ̀ȅ́ȇ̋>.[27] The IPA also proposes the option of tone letters, e.g. <˩˥˦˧˨>, which are much less often used, but are more consistent for contours. Tone letters in the IPA have five different levels, and sequences of these letters can be used to indicate contours. Well-designed fonts will even merge a sequence of tone letters into a contour. For example, compare the font Linux Libertine, which does not merge tone letters <˩˧˦˧˨>, with the font CharisSIL, which merges this sequence of four tone letters into a single contour <˩˧˨>. For strict IPA encoding we propose to standardize on tone letters.

[27] To make things even more complex, there are at least two different Unicode homoglyphs for the low and high level tones, namely <◌̀> COMBINING GRAVE TONE MARK at U+0340 vs. <◌̀> COMBINING GRAVE ACCENT at U+0300 for low tone, and <◌́> COMBINING ACUTE TONE MARK at U+0341 vs.<◌́> COMBINING ACUTE ACCENT at U+0301 for high tone.

Second, we commonly encounter the use of <g> LATIN SMALL LETTER G at U+0067, instead of the Unicode Standard IPA character for the voiced velar stop <ɡ> LATIN SMALL LETTER SCRIPT G at U+0261. One begins to question whether this issue is at all apparent to the working linguist, or if they simply use the U+0067 because it is easily keyboarded and thus saves time, whereas the latter must be cumbersomely inserted as a special symbol in most software. The International Phonetic Association has taken the stance that both the keyboard LATIN SMALL LETTER G and the LATIN SMALL LETTER SCRIPT G are valid input characters for the voiced velar plosive. Unfortunately, this decision further introduces ambiguity for linguists trying to adhere to a strict Unicode Standard IPA encoding. For strict IPA encoding we propose to standardize on the more idiosyncratic LATIN SMALL LETTER SCRIPT G at U+0261.

Third, the IPA has special markers for velarization <◌ˠ> and pharyngealization <◌ˤ>. Confusingly, there is also a marker for "velarized or pharyngealized", using the <◌̴> COMBINING TILDE OVERLAY at U+0334. The tilde overlay seems to be extremely rarely used. We suggest to try and avoid using the tilde overlay, though for reasons of backward compatibility we will allow it in valid-IPA.

Finally, the IPA states that "diacritics may be placed above a symbol with a descender". For example, for marking marking of voiceless pronunciation of voiced segments the IPA uses the ring diacritic. Originally, the ring should be placed below the base character, like in <m̥>, using the COMBINING RING BELOW at U+0325. However, in letters with long descenders the IPA also allows to put the ring above the base, like in <n̊>, using the COMBINING RING ABOVE at U+030A. Yet, proper font design does not have any problem with rendering the ring below the base character, like in <ŋ̊>, so for strict IPA encoding we propose to standardize on the ring below. As a principle, for strict IPA encoding only one options should be allowed for all diacritics.

The variable encoding as allowed by the IPA becomes even more troublesome for the tilde and diaeresis diacritics. In these cases, the IPA itself attaches different semantics to the symbols above and below a base characters. The tilde above a character (like in <ã>, using the COMBINING TILDE at U+0303) is used for nasalization, while the tilde below a character (like in <a̰>, using the COMBINING TILDE BELOW at U+0330) indicates creaky voice. Likewise, the diaeresis above (like in <ä>, using the COMBINING DIAERESIS at U+0308) is used for centralization, while the diaeresis below a character (like in <a̤>, using the COMBINING DIAERESIS BELOW at U+0324) indicates breathy voice. These cases strengthen our plea to not allow diacritics to switch position for typographic convenience.

3.7 Pitfall: Tie bar

In the major revision of the IPA in 1932, affricates were represented by two consonants <tʃ>, ligatures <ʧ>, or with the tie-bar <t͡ʃ>. In the 1938 revision the tie-bar's semantics were broadened to indicate simultaneous articulation, as for example in labial velars such as <k͡p>. Thus, the tie-bar is a convenient diacritic for visually tokenizing input strings into chunks of phonetically salient groups, including affricates, doubly articulated consonants or diphthongs.

The tie bar can be placed above or below the base characters, e.g. <t͡s> or <t͜s>. IPA allows both options. The choice between the two symbols is purely for legible rendering; there is no difference in semantics between the two symbols. However, rendering is such a problematic issue for tie bars in general that many linguists simply do not use them. Just looking at a few different fonts already indicates that actually no font designer really gets the placement right in combination with superscripts and subscripts. If really necessary, we propose to standardize on the tie bar above the base characters, using a COMBINING DOUBLE INVERTED BREVE at U+0361.[28]

Times new Roman: t̥ʰ͡s t̥ʰ͜s

CharisSIL: t̥ʰ͡s t̥ʰ͜s

Monaco: t̥ʰ͡s t̥ʰ͜s

DoulosSIL: t̥ʰ͡s t̥ʰ͜s

Linux Libertine: t̥ʰ͡s t̥ʰ͜s

Tie bars are a special type of character in the sense that they do not belong to a segment, but bind two graphemes together. This actually turns out to be rather different from Unicode conceptions. The Unicode encoding of this character belong to the Combining Diacritical Marks, namely either COMBINING DOUBLE INVERTED BREVE at U+0361 or COMBINING DOUBLE BREVE BELOW at U+035C. Such a combining mark is by definition tied to the character in front, but not the character following it. The Unicode treatment of this character thus only partly corresponds to the IPA conception, which ideally would have the tie bar linked both to the character in front and to the character following.

[28] Also note that the UNDERTIE at U+203F looks like the tie bar below and is easily confused with it. However, it is a different character and has a different function in IPA. The undertie is used as a linking symbol to indicate the lack of a boundary, e.g. French *petit ami* [pətit‿ami] 'boyfriend'.

Further, according to the spirit of the IPA, it would also be possible to combine more than two base characters into one tie bar, but this is not possible with Unicode (i.e. there is no possibility to draw a tie bar over three of four characters). It is possible to indicate such larger groups by repeating the tie bar, like for a triphthong <a͡ʊ͡ə> in the English word *hour*. If really necessary, we consider this possible, even though the rendering will never look good.

Most importantly though, in comparison to normal Unicode processing, the tie-bar actually takes a reversed approach to complex graphemes. Basically, the Unicode principle (see Section 2.3) is that fixed sequences in a writing system have to be specified as "tailored" grapheme clusters. In case the sequence is not a cluster, this has to be explicitly indicated. IPA takes a different approach. In IPA by default different base letters are not connected into larger clusters; only when it is specified in the string itself (using the tie bar).

3.8 Pitfall: Ligatures and digraphs

One important distinction to acknowledge is the difference between multigraphs and ligatures. Multigraphs are groups of characters (in the context of IPA e.g. <tʃ> or <ou>) while ligatures are single characters (e.g. <ʧ> LATIN SMALL LETTER TESH DIGRAPH at U+02A7). Ligatures arose in the context of printing easier-to-read texts, and are included in the Unicode Standard for reasons of legacy encoding. However, their usage is discouraged by the Unicode core specification. Specifically related to IPA, various phonetic combinations of characters (typically affricates) are available as single code-points in the Unicode Standard, but are designated DIGRAPHS. Such glyphs might be used by software to produce a pleasing display, but they should not be hard-coded into the text itself. In the context of IPA, characters like the following ligatures should thus *not* be used. Instead a combination of two characters is preferred:

<ʣ> LATIN SMALL LETTER DZ DIGRAPH at U+02A3 (use <dz>)
<ʤ> LATIN SMALL LETTER DEZH DIGRAPH at U+02A4 (use <dʒ>)
<ʥ> LATIN SMALL LETTER DZ DIGRAPH WITH CURL at U+02A5 (use <dʑ>)
<ʦ> LATIN SMALL LETTER TS DIGRAPH at U+02A6 (use <ts>)
<ʧ> LATIN SMALL LETTER TESH DIGRAPH at U+02A7 (use <tʃ>)
<ʨ> LATIN SMALL LETTER TC DIGRAPH WITH CURL at U+02A8 (use <tɕ>)
<ʩ> LATIN SMALL LETTER FENG DIGRAPH at U+02A9 (use <fŋ>)

However, there are a few Unicode characters that are historically ligatures, but which are today considered as simple characters in the Unicode Standard and thus should be used when writing IPA, namely:

<ɮ> LATIN SMALL LETTER LEZH at U+026E
<œ> LATIN SMALL LIGATURE OE at U+0153
<ɶ> LATIN LETTER SMALL CAPITAL OE at U+0276
<æ> LATIN SMALL LETTER AE at U+00E6

3.9 Pitfall: Missing decomposition

Although many combinations of base character with diacritic are treated as canonical equivalent with precomposed characters, there are a few combinations in IPA that allow for multiple, apparently identical, encodings that are not canonical equivalent (see Section 2.9). The following elements should not be treated as diacritics when encoding IPA in Unicode:

<◌̡> COMBINING PALATALIZED HOOK BELOW at U+0321
<◌̢> COMBINING RETROFLEX HOOK BELOW at U+0322
<◌̵> COMBINING SHORT STROKE OVERLAY at U+0335
<◌̸> COMBINING SHORT SOLIDUS OVERLAY at U+0337

There turn out to be a lot of characters in the IPA that could be conceived as using any of these elements, like <ɲ>, <ɳ>, <ɨ> or <ø>. However, all such characters exist as well as precomposed combination in Unicode, and these precomposed characters should preferably be used. When instead combinations of a base character with diacritic are used, then these combinations are not canonical equivalent to the precomposed combinations. This means that any search will not find both at the same time.

A similar problem arises with the rhotic hook. There are two precomposed characters in Unicode with a rhotic hook, which are not canonical equivalent with a combination of the vowel with a separately encoded hook:

<ɚ> LATIN SMALL LETTER SCHWA WITH HOOK at U+025A
<ɝ> LATIN SMALL LETTER REVERSED OPEN E WITH HOOK at U+025D

All other combinations of vowels with rhotic hooks will have to be made by using <◌˞> MODIFIER LETTER RHOTIC HOOK at U+02DE, because there is no complete set of precomposed characters with rhotic hooks in Unicode. For that reason

we propose to not use the two precomposed characters with hooks mentioned above, but always use the separate rhotic hook at U+02DE in IPA.

A similar situation arises with <◌̴> COMBINING TILDE OVERLAY at U+0334. The main reason some phoneticians like to use this in IPA is to mark the "dark" <l> in English codas, using the character <ɬ> LATIN SMALL LETTER L WITH MIDDLE TILDE at U+026B. This character is not canonically equivalent to the combination <l> + <◌̴>, so one of the two possible encodings has to be chosen. Because the tilde overlay is described as a general mechanism by the IPA, we propose to use the separated <◌̴> COMBINING TILDE OVERLAY at U+0334. However, note that phonetically this seems to be (almost) superfluous (see Section 3.6) and the typical usage in the form of <ɬ> is (almost) a homoglyph with <ɬ> (see Section 3.5). For these reasons we also suggest to try and avoid the tilde overlay completely.

Reversely, note that the <ç> LATIN SMALL LETTER C WITH CEDILLA at U+00E7 is canonically equivalent with <c> with <◌̧> COMBINING CEDILLA at U+0327, so it will be separated into two characters by Unicode canonical decomposition, also if such a decomposition is not intended in the IPA. However, because of the nature of canonical equivalence (see Section 2.9), these two encodings are completely identical in any computational treatment, so this decomposition does not have any practical consequences.

3.10 Pitfall: Different notions of diacritics

Another pitfall relates to the question of what are diacritics. The problem is that the meaning of the term diacritics as used by the IPA is not the same as is used in the Unicode Standard. Specifically, diacritics in the IPA-sense are either so-called COMBINING DIACRITICAL MARKS or SPACING MODIFIER LETTERS in the Unicode Standard. Crucially, Combining Diacritical Marks are by definition combined with the character before them (to form so-called default grapheme clusters, see Section 1.4). In contrast, Spacing Modifier Letters are by definition *not* combined into grapheme clusters with the preceding character, but simply treated as separate letters. In the context of the IPA, the following IPA-diacritics are actually Spacing Modifier Letters in the Unicode Standard:

Length marks, namely:

<◌ː> MODIFIER LETTER TRIANGULAR COLON at U+02D0
<◌ˑ> MODIFIER LETTER HALF TRIANGULAR COLON at U+02D1

Tone letters, like:

<˥> MODIFIER LETTER EXTRA-HIGH TONE BAR at U+02E5
<˩> MODIFIER LETTER LOW TONE BAR at U+02E8
and others like this

Superscript letters, like:

<◌ʰ> MODIFIER LETTER SMALL H at U+02B0
<◌ˤ> MODIFIER LETTER SMALL REVERSED GLOTTAL STOP at U+02E4
<◌ⁿ> SUPERSCRIPT LATIN SMALL LETTER N at U+207F
and many more like this

The rhotic hook:[29]

<◌˞> MODIFIER LETTER RHOTIC HOOK at U+02DE

Although linguists might expect these characters to belong together with the character in front of them, at least for <ʰ> MODIFIER LETTER SMALL H at U+02B0 the Unicode Consortium's decision to treat it as a separate character is also linguistically correct, because according to the IPA it can be used both for post-aspiration (following the base character) and pre-aspiration (preceding the base character). The default combination of Spacing Modifiers with the preceding character can be specified in orthography profiles (see Chapter 4).

3.11 Pitfall: No unique diacritic ordering

Also related to diacritics is the question of ordering. To our knowledge, the International Phonetic Association does not specify a specific ordering for diacritics that combine with phonetic base symbols; this exercise is left to the reasoning of the transcriber. However, such marks have to be explicitly ordered if sequences of them are to be interoperable and compatible computationally. An example is a labialized aspirated alveolar plosive: <tʷʰ>. There is nothing holding linguists back from using <tʰʷ> instead (with exactly the same intended meaning). However, from a technical standpoint, these two sequences are different, e.g. if both sequences are used in a document, searching for <tʷʰ> will not find any instances of <tʰʷ>, and vice versa. Likewise, a creaky voiced syllabic dental nasal can be encoded in various orders, e.g. <n̰̪̩>, <n̪̰̩> or <n̩̪̰>.

[29] It is really unfortunate that the rhotic hook in Unicode is classified as a Spacing Modifier, and not as a Combining Diacritical Mark. Although the rhotic hook is placed to the right of its base character (and not above or below), it still is always connected to the character in front, even physically connected to it. We cannot find any reason for this treatment, and consider it an error in Unicode. We hope it will be possible to change this classification in the future.

Canonical combining classes

In accordance with the absence of any specification of ordering in the IPA, the Unicode Standard likewise does not propose any standardized orders. Both leave it to the user to be consistent; this approach naturally invites inconsistency across different authored resources.

There is one (minor) aspect of ordering for which the Unicode Standard does present a canonical solution. Fortunately, this is uncontroversial from a linguistic perspective. Diacritics in the Unicode Standard (i.e. Combining Diacritical Marks, see Section 3.10) are classified in so-called CANONICAL COMBINING CLASSES. In practice, the diacritics are distinguished by their position relative to the base character.[30] When applying a Unicode normalization (NFC or NFD, see Section 2.9), the diacritics in different positions are put in a specified order. This process therefore harmonizes the difference between different encodings in some situations, for example in the case of an extra-short creaky voice vowel <ĕ̯>. This grapheme cluster can be encoded either as <e>+<◌̆>+<◌̯> or as <e>+<◌̯>+<◌̆>. To prevent this twofold encoding, the Unicode Standard specifies the second ordering as canonical (namely, diacritics below are put before diacritics above).

When encoding a string according to the Unicode Standard, it is possible to do this either using the NFC (composition) or NFD (decomposition) normalization (see Section 2.9). Decomposition implies that precomposed characters (like <á> LATIN SMALL LETTER A WITH ACUTE at U+00E1) will be split into its parts. This might sound preferable for a linguistic analysis, as the different diacritics are separated from the base characters. However, note that most attached elements like strokes (e.g. in the <ɨ>), retroflex hooks (e.g. in <ʐ>) or rhotic hooks (e.g. in <ɜ˞>) will not be decomposed, but strangely enough a cedilla (like in <ç>) will be decomposed (see Section 3.9). In general, Unicode decomposition does not behave like a feature decomposition as expected from a linguistic perspective. It is thus important to consider Unicode decomposition only as a technical procedure, and not assume that it is linguistically sensible.

Proposal for diacritic ordering

Facing the problem of specifying a consistent ordering of diacritics while developing a large database of phonological inventories from the world's languages, Moran (2012: 540) defined a set of diacritic ordering conventions. The conventions are influenced by the linguistic literature, though some ad-hoc decisions

[30] See http://unicode.org/reports/tr44/#Canonical_Combining_Class_Values for a detailed description.

had to be taken given the vast variability of phonological segments described by linguists.

By Unicode Canonical Combining Classes, the diacritics on top of a character, like <◌⃪> (Combining Class number 1), always come before diacritics below (Combining Class number 220), which in turn always come before diacritics above (Combining Class number 230), which in turn come before diacritics over multiple characters like the tie bar <◌͡◌> (Combining Class number 233). We follow this order, but add the other IPA diacritics (which are not diacritics in the Unicode sense) between diacritics below and the tie bar. Further, within all these classes of diacritics there is no canonical ordering specified by Unicode, so we propose an explicit ordering here.

Starting with the diacritics below: if a character sequence contains more than one diacritic below the base character, then the place features are applied first (linguolabial, dental, apical, laminal, advanced, retracted), followed by the manner features (raised, lowered, advanced and retracted tongue root), then secondary articulations (more round, less round), laryngeal settings (creaky, breathy, voiced, devoiced), and finally the syllabic or non-syllabic marker. So, the order that is proposed is the following, where <|> indicates *or* and <⟶> indicates *precedes*. Note that the groups of alternatives (as marked by <|>) are supposed never to occur together with the same base character. In effect, this represents yet another restriction on possible diacritic sequences.

Combining Diacritical Marks (below) ordering:

⟶ linguolabial <◌̼> | dental <◌̪> | apical <◌̺> | laminal <◌̻>
⟶ advanced <◌̟> | retracted <◌̠>
⟶ raised <◌̝> | lowered <◌̞>
⟶ advanced tongue root <◌̘> | retracted tongue root <◌̙>
⟶ more rounded <◌̹> | less rounded <◌̜>
⟶ creaky voiced <◌̰> | breathy voiced <◌̤> | voiced <◌̬> | voiceless <◌̥>
⟶ syllabic <◌̩> | non-syllabic <◌̯>

Next, if a character sequence contains more than one diacritic above the base character, we propose the following order:

Combining Diacritical Marks (above) ordering:

⟶ nasalized <◌̃>
⟶ centralized <◌̈> | mid-centralized <◌̽>
⟶ extra short <◌̆>

→ no audible release <◌̚>

Then, when a character sequence contains more than one character of the Spacing Modifier Letters, these will be placed after all combining diacritic marks in the following order:

Spacing Modifier Letters ordering:

→ rhotic hook <◌˞>
→ lateral release <◌ˡ> | nasal release <◌ⁿ>
→ labialized <◌ʷ>
→ palatalized <◌ʲ>
→ velarized <◌ˠ>
→ pharyngealized <◌ˤ>
→ aspirated <◌ʰ> | ejective <◌ʼ>
→ long <◌ː> | half-long <◌ˑ>

Finally, the tie bar follows at the very end of any such sequence:

Tie bar:

→ tie bar <◌͡◌>

3.12 Pitfall: Future revisions to the IPA

With each revision of the IPA, many decisions need to be made by the Association as to which symbols should be added, removed or changed. For example, in the 1989 revision of the IPA at the Kiel Convention, changes to specific symbols (in previous charts) were debated and the Association's members certain decisions. The prevailing mood at the convention was not to change specific symbols unless a strong case was made Ladefoged 1990. For example, two such decisions included:

- symbols for clicks were changed from <ɻ ɮ ƈ> to <ǀ ǁ !> because the latter were the symbols used by nearly all Khoisanists and Bantuists.
- the Americanist tradition of using using <◌̌, a COMBINING CARON at U+030C for all post-alveolar sounds, like in <š ž č ǰ>, was not adopted because the Association members at the convention "were not sufficiently impressed by arguments … to the effect that these sounds formed a natural class, and thus is would be appropriate to recognize this by maintaining a common aspect to their symbolism" (Ladefoged 1990: 62).

These decisions have practical consequences for transcribers of IPA, particularly those who wish to follow recommended practices of encoding electronic text in the Unicode Standard. For example, the Unicode Standard contains removed symbols as labels them as clicks, e.g. <ɻ> LATIN SMALL LETTER TURNED T at U+0287 in the IPA EXTENSIONS block has the comment 'dental click (sound of "tsk tsk")'. In this case, the IPA transcriber must know the status of click symbols in the current version of the IPA and then identify those characters within the Unicode Standard.

The most controversial issue regarding symbols debated at the convention was the representation for voiceless implosives Ladefoged 1990: 62. Recall the principles of the IPA outlined in Section 3.2, such as using distinct symbols for phonological contrast versus convenience of display in the chart, must be taken into account when arguing for or against the inclusion or deletion of IPA symbols in the IPA chart. Furthermore, the inclusion of deletion of symbols should incorporate the phonetic knowledge of the world's languages.

Ladefoged 1990 argued against the inclusion of these symbols < ɓ, ƭ, ƈ, ƙ, ƈ͡ʃ> voiceless implosives, noting that: they are not contrastive (e.g. in Mayan languages); there is no instrumental evidence supporting voiceless implosives in Africa; the sounds are sufficiently rare as to not need a whole new row of symbols in the chart. Ladefoged favored symbolizing the sounds using a voiceless diacritic ring below voiced implosives, e.g. <ɓ̥>.

Nevertheless, in the 1989 IPA chart there is indeed a row for implosives containing voiceless and voiced pairs.[31] But already in the next revision, in 1993 (with an update in 1996), the voiceless implosives were dropped. The implosives row from the IPA consonantal chart disappeared and voiced implosives were given a column in the non-pulmonic consonants table (which is still reflected in the latest revision to date, IPA 2005).

The Journal of the International Phonetic Association follows its own published standard for the IPA at the time of publication, even when it may conflict with the Association's principle of using different symbols for contrastive sounds and diacritics for phonetic variation. For example, in the case of voiceless implosives, Mc Laughlin 2005 shows that Seereer-Siin (Niger-Congo; Atlantic; ISO 639-3: srr) has a phonologically contrastive set of voiced and voiceless implosive

[31] https://en.wikipedia.org/wiki/File:IPA_as_of_1989.png

stops at the labial, coronal and palatal places of articulation. These symbols are transcribed in an *Illustrations of the IPA* article in the IPA journal as < ɓ̥, ɗ̥, ʄ̥ >.[32]

The point of this pitfall is to highlight that revisions to the IPA will continue into the future, albeit they are infrequent. Nevertheless, given the Unicode Standard's principle of maintaining backwards compatibility (at all costs), transcribers and consumers of IPA cannot rely solely on remarks in the Unicode Standard to reflect current standard IPA usage. There is the possibility that at a later revision of the IPA, symbols that are not currently encoded in the Unicode Standard are added to the IPA – although we think this is unlikely.

3.13 Additions to the IPA

In the course of collecting a large sample of phoneme systems across the world's languages (Moran 2012), we found that in order to preserve distinctions both within and across language descriptions, additions to the approved IPA glyph set were needed. Wherever possible these additions were drawn from the extIPA symbols for disordered speech.[33] This section describes our proposed additions to the IPA glyph set. These additions are not part of the official IPA recommendations, so they should be used with care.

- Retroflex click
  Retroflex clicks can be represented by <‼> double exclamation mark at U+203C. Note that the (post-)alveolar click <!> at U+01C3 is confusingly referred to as latin letter retroflex click in the Unicode standard, which is probably best seen as an error.
- Voiced retroflex implosive
  Although the IPA includes a series of voiced implosives (marked with a hook on top, see Section 3.9), there is no voiced retroflex implosive. Following the spirit of the IPA, we propose to use <ᶑ> latin small letter d with hook and tail at U+1D91 for this sound.
- Fortis/lenis
  Languages described as having a fortis/plain/lenis distinction that corresponds poorly with the traditional voiced/voiceless-unaspirated/voiceless-aspirated continuum can be marked using the voiceless glyph for the plain

[32] Note that the IPA chart states that diacritics can also be placed above a symbol, e.g. reportedly for legibility's sake < ɡ̊ > vs < g̊ >. This of course leads to the variability in encoding that we are trying to address on a computational level in this work. See Section **??**.

[33] **Add better reference here**: https://www.internationalphoneticassociation.org/sites/default/files/extIPAChart2008.pdf

phoneme, and then <◌͈> COMBINING DOUBLE VERTICAL LINE BELOW at U+0348 to mark the fortis articulation, and/or <◌͉> COMBINING LEFT ANGLE BELOW at U+0349 for the lenis articulation.

- FRICTIONALIZATION
  The diacritic <◌̻> COMBINING X BELOW at U+0353 can be used to represent three types of frictionalized sounds: First, click consonants where the release of the anterior closure involves an ingressive "sucking" sound similar to a fricative, for example <k!̻ʰ>; second, "frictionalized" vowels (sounds that are phonologically vocalic, but with sufficiently close closures to create buzzing); and third, fricative sounds at places of articulation that do not have dedicated fricative glyphs, for example sounds with voiceless velar lateral frication, like <ʟ̻>.
- DERHOTICIZATION
  For derhoticization we propose to use <◌̯> COMBINING BREVE BELOW at U+032E. **WHAT EXACTLY IS THIS?**
- CORONAL NON-SIBILANT
  Languages described as having a sibilant/non-sibilant distinction among coronal fricatives and affricates can be handled using the subscript <◌̼> COMBINING EQUALS SIGN BELOW at U+0347 to mark the non-sibilant phoneme.
- GLOTTALIZATION
  Glottalized sounds can be indicated using <◌ˀ> MODIFIER LETTER GLOTTAL STOP at U+02C0, unless it is clear that either "ejective" or "creaky voicing" are the intended sounds (in which cases the standard IPA diacritics should be used). Pre-glottalized sounds can be marked with <ˀ◌> to the left of the base glyph, for example <ˀt>.
- VOICED PRE-ASPIRATION
  Voiced sounds having pre-aspiration can be marked with <ʱ◌> MODIFIER LETTER SMALL H WITH HOOK at U+02B1 to the left of the base glyph, for example <ʱd>.
- EPILARYNGEAL PHONATION
  There are some rare articulations that make use of an epilaryngeal phonation mechanism (e.g., the "sphincteric vowels" of !Xóõ). To represent these vowels, we propose to use the modifier <◌ᴱ> MODIFIER LETTER CAPITAL E at U+1D31 to denote such sphincteric phonation.

3.14 Recommendations

Summarizing the pitfalls as discussed in this chapter, we propose to define three different IPA encodings: strict-IPA, valid-IPA and widened-IPA. These three encodings are subsets of each other, i.e. strict-IPA is more restricted than valid-IPA, which it turn is more restricted than widened-IPA. Informally speaking, valid-IPA represents the current state of the IPA (International Phonetic Association 2005). Strict-IPA represents a more constrained version of IPA, while widened-IPA is a slightly extended version of IPA, allowing a few more symbols.

Strict-IPA encoding is supposed to be used when interoperability of phonetic resources is intended. It is a strongly constrained subset of IPA geared towards uniqueness of encoding. Ideally, for each transcription there should be exactly one possible strict-IPA encoding. For each phonetic feature there is only one possibility (see Section 3.6) and the IPA diacritics are forced into a canonical ordering (see Section 3.11).

Valid-IPA does allow alternative symbols with the same phonetic meaning, as specified in the official IPA specifications. Also, valid-IPA does not enforce a specific ordering of diacritics, because the IPA does not propose any such ordering. This means that in valid-IPA the same phonetic intention can be encoded in multiple ways. This is sufficient for phonetically trained human eyes, but it is not sufficient for automatic interoperability.

Finally, widened-IPA includes a few more symbols which seem to be useful for various special cases (see Section 3.13).

At the end of this chapter we have added a few longish tables summarizing all 159 different unicode codepoints that form the basis of strict-IPA encoding (107 letters, 36 diacritics and 16 remaining symbols). Each of these tables shows a typical glyph, and then lists the Unicode Codepoint, Unicode Name and IPA description for each symbol. Further, there is a table with the additional options for valid-IPA and a table with the additional options for widened-IPA.

textbfNOTE: I HAVE ADDED VARIOUS TONE DIACRITICS TO VALID-IPA THAT ARE NOT IN THE OFFICIAL IPA SUMMARY. SHOULD THEY BE INCLUDED?

- STRICT-IPA LETTERS
  The 107 different IPA letters as allowed in strict-IPA encoding are listed in Table 3.1 starting on page 67.

- STRICT-IPA DIACRITICS
  The 36 different IPA diacritics and tone markers (both Unicode Modifier Letters and Combining Diacritical Marks) as allowed in strict-IPA encoding are listed in Table 3.2 starting on page 70.
- STRICT-IPA REMAINDERS
  The 16 remaining IPA symbols (boundary, stress, tone letters and intonation markers) as allowed in strict-IPA encoding are listed in Table 3.3 on page 71.
- VALID-IPA ADDITIONS
  The 16 additional symbols as allowed in valid-IPA encoding are listed in Table 3.4 on page 72.
- WIDENED-IPA ADDITIONS
  The 10 proposed additions to the IPA are listed in Table 3.5 on page 73.

Table 3.1: Strict-IPA letters with Unicode encodings

	Code	Unicode name	IPA name
a	U+0061	LATIN SMALL LETTER A	open front unrounded
æ	U+00E6	LATIN SMALL LETTER AE	raised open front unrounded
ɐ	U+0250	LATIN SMALL LETTER TURNED A	lowered schwa
ɑ	U+0251	LATIN SMALL LETTER ALPHA	open back unrounded
ɒ	U+0252	LATIN SMALL LETTER TURNED ALPHA	open back rounded
b	U+0062	LATIN SMALL LETTER B	voiced bilabial plosive
ʙ	U+0299	LATIN LETTER SMALL CAPITAL B	voiced bilabial trill
ɓ	U+0253	LATIN SMALL LETTER B WITH HOOK	voiced bilabial implosive
c	U+0063	LATIN SMALL LETTER C	voiceless palatal plosive
ç	U+00E7	LATIN SMALL LETTER C WITH CEDILLA	voiceless palatal fricative
ɕ	U+0255	LATIN SMALL LETTER C WITH CURL	voiceless alveolo-palatal fricative
d	U+0064	LATIN SMALL LETTER D	voiced alveolar plosive
ð	U+00F0	LATIN SMALL LETTER ETH	voiced dental fricative
ɖ	U+0256	LATIN SMALL LETTER D WITH TAIL	voiced retroflex plosive
ɗ	U+0257	LATIN SMALL LETTER D WITH HOOK	voiced dental/alveolar implosive
e	U+0065	LATIN SMALL LETTER E	close-mid front unrounded
ə	U+0259	LATIN SMALL LETTER SCHWA	mid-central schwa
ɛ	U+025B	LATIN SMALL LETTER OPEN E	open-mid front unrounded
ɘ	U+0258	LATIN SMALL LETTER REVERSED E	close-mid central unrounded
ɜ	U+025C	LATIN SMALL LETTER REVERSED OPEN E	open-mid central unrounded
ɞ	U+025E	LATIN SMALL LETTER CLOSED REVERSED OPEN E	open-mid central rounded
f	U+0066	LATIN SMALL LETTER F	voiceless labiodental fricative
g	U+0261	LATIN SMALL LETTER SCRIPT G	voiced velar plosive
ɢ	U+0262	LATIN LETTER SMALL CAPITAL G	voiced uvular plosive
ɠ	U+0260	LATIN SMALL LETTER G WITH HOOK	voiced velar implosive
ɢ̆	U+029B	LATIN LETTER SMALL CAPITAL G WITH HOOK	voiced uvular implosive
ɤ	U+0264	LATIN SMALL LETTER RAMS HORN	close-mid back unrounded
ɣ	U+0263	LATIN SMALL LETTER GAMMA	voiced velar fricative
h	U+0068	LATIN SMALL LETTER H	voiceless glottal fricative
ħ	U+0127	LATIN SMALL LETTER H WITH STROKE	voiceless pharyngeal fricative
ʜ	U+029C	LATIN LETTER SMALL CAPITAL H	voiceless epiglottal fricative

continued on next page

Table 3.1 Strict-IPA letters with Unicode encodings — *continued*

	Code	Unicode name	IPA name
ɦ	U+0266	LATIN SMALL LETTER H WITH HOOK	voiced glottal fricative
ɧ	U+0267	LATIN SMALL LETTER HENG WITH HOOK	simultaneous voiceless postalveolar+velar fricative
ɥ	U+0265	LATIN SMALL LETTER TURNED H	voiced labial-palatal approximant
i	U+0069	LATIN SMALL LETTER I	close front unrounded
ɪ	U+026A	LATIN LETTER SMALL CAPITAL I	lax close front unrounded
ɨ	U+0268	LATIN SMALL LETTER I WITH STROKE	close central unrounded
j	U+006A	LATIN SMALL LETTER J	voiced palatal approximant
ʝ	U+029D	LATIN SMALL LETTER J WITH CROSSED TAIL	voiced palatal fricative
ɟ	U+025F	LATIN SMALL LETTER DOTLESS J WITH STROKE	voiced palatal plosive
ʄ	U+0284	LATIN SMALL LETTER DOTLESS J WITH STROKE AND HOOK	voiced palatal implosive
k	U+006B	LATIN SMALL LETTER K	voiceless velar plosive
l	U+006C	LATIN SMALL LETTER L	voiced alveolar lateral approximant
ʟ	U+029F	LATIN LETTER SMALL CAPITAL L	voiced velar lateral approximant
ɬ	U+026C	LATIN SMALL LETTER L WITH BELT	voiceless alveolar lateral fricative
ɭ	U+026D	LATIN SMALL LETTER L WITH RETROFLEX HOOK	voiced retroflex lateral approximant
ɮ	U+026E	LATIN SMALL LETTER LEZH	voiced alveolar lateral fricative
ʎ	U+028E	LATIN SMALL LETTER TURNED Y	voiced palatal lateral approximant
m	U+006D	LATIN SMALL LETTER M	voiced bilabial nasal
ɱ	U+0271	LATIN SMALL LETTER M WITH HOOK	voiced labiodental nasal
n	U+006E	LATIN SMALL LETTER N	voiced alveolar nasal
ɴ	U+0274	LATIN LETTER SMALL CAPITAL N	voiced uvular nasal
ɲ	U+0272	LATIN SMALL LETTER N WITH LEFT HOOK	voiced palatal nasal
ɳ	U+0273	LATIN SMALL LETTER N WITH RETROFLEX HOOK	voiced retroflex nasal
ŋ	U+014B	LATIN SMALL LETTER ENG	voiced velar nasal
o	U+006F	LATIN SMALL LETTER O	close-mid back rounded
ø	U+00F8	LATIN SMALL LETTER O WITH STROKE	close-mid front rounded
œ	U+0153	LATIN SMALL LIGATURE OE	open-mid front rounded
ɶ	U+0276	LATIN LETTER SMALL CAPITAL OE	open front rounded
ɔ	U+0254	LATIN SMALL LETTER OPEN O	open-mid back rounded
ɵ	U+0275	LATIN SMALL LETTER BARRED O	close-mid central rounded

continued on next page

Table 3.1 Strict-IPA letters with Unicode encodings — *continued*

	Code	Unicode name	IPA name
p	U+0070	LATIN SMALL LETTER P	voiceless bilabial plosive
ɸ	U+0278	LATIN SMALL LETTER PHI	voiceless bilabial fricative
q	U+0071	LATIN SMALL LETTER Q	voiceless uvular plosive
r	U+0072	LATIN SMALL LETTER R	voiced alveolar trill
ʀ	U+0280	LATIN LETTER SMALL CAPITAL R	voiced uvular trill
ɹ	U+0279	LATIN SMALL LETTER TURNED R	voiced alveolar approximant
ɺ	U+027A	LATIN SMALL LETTER TURNED R WITH LONG LEG	voiced alveolar lateral flap
ɻ	U+027B	LATIN SMALL LETTER TURNED R WITH HOOK	voiced retroflex approximant
ɽ	U+027D	LATIN SMALL LETTER R WITH TAIL	voiced retroflex tap
ɾ	U+027E	LATIN SMALL LETTER R WITH FISHHOOK	voiced alveolar tap
ʁ	U+0281	LATIN LETTER SMALL CAPITAL INVERTED R	voiced uvular fricative
s	U+0073	LATIN SMALL LETTER S	voiceless alveolar fricative
ʂ	U+0282	LATIN SMALL LETTER S WITH HOOK	voiceless retroflex fricative
ʃ	U+0283	LATIN SMALL LETTER ESH	voiceless postalveolar fricative
t	U+0074	LATIN SMALL LETTER T	voiceless alveolar plosive
ʈ	U+0288	LATIN SMALL LETTER T WITH RETROFLEX HOOK	voiceless retroflex plosive
u	U+0075	LATIN SMALL LETTER U	close back rounded
ʉ	U+0289	LATIN SMALL LETTER U BAR	close central rounded
ɯ	U+026F	LATIN SMALL LETTER TURNED M	close back unrounded
ɰ	U+0270	LATIN SMALL LETTER TURNED M WITH LONG LEG	voiced velar approximant
ʊ	U+028A	LATIN SMALL LETTER UPSILON	lax close back rounded
v	U+0076	LATIN SMALL LETTER V	voiced labiodental fricative
ʋ	U+028B	LATIN SMALL LETTER V WITH HOOK	voiced labiodental approximant
ⱱ	U+2C71	LATIN SMALL LETTER V WITH RIGHT HOOK	voiced labiodental tap
ʌ	U+028C	LATIN SMALL LETTER TURNED V	open-mid back unrounded
w	U+0077	LATIN SMALL LETTER W	voiced labial-velar approximant
ʍ	U+028D	LATIN SMALL LETTER TURNED W	voiceless labial-velar fricative
x	U+0078	LATIN SMALL LETTER X	voiceless velar fricative
y	U+0079	LATIN SMALL LETTER Y	close front rounded
ʏ	U+028F	LATIN LETTER SMALL CAPITAL Y	lax close front rounded
z	U+007A	LATIN SMALL LETTER Z	voiced alveolar fricative
ʐ	U+0290	LATIN SMALL LETTER Z WITH RETROFLEX HOOK	voiced retroflex fricative

continued on next page

Table 3.1 Strict-IPA letters with Unicode encodings — *continued*

	Code	Unicode name	IPA name
ʑ	U+0291	LATIN SMALL LETTER Z WITH CURL	voiced alveolo-palatal fricative
ʒ	U+0292	LATIN SMALL LETTER EZH	voiced postalveolar fricative
ʔ	U+0294	LATIN LETTER GLOTTAL STOP	voiceless glottal plosive
ʕ	U+0295	LATIN LETTER PHARYNGEAL VOICED FRICATIVE	voiced pharyngeal fricative
ʡ	U+02A1	LATIN LETTER GLOTTAL STOP WITH STROKE	epiglottal plosive
ʢ	U+02A2	LATIN LETTER REVERSED GLOTTAL STOP WITH STROKE	voiced epiglottal fricative
ǀ	U+01C0	LATIN LETTER DENTAL CLICK	voiceless dental click
ǁ	U+01C1	LATIN LETTER LATERAL CLICK	voiceless alveolar lateral click
ǂ	U+01C2	LATIN LETTER ALVEOLAR CLICK	voiceless palatoalveolar click
!	U+01C3	LATIN LETTER RETROFLEX CLICK	voiceless (post)alveolar click
ʘ	U+0298	LATIN LETTER BILABIAL CLICK	voiceless bilabial click
β	U+03B2	GREEK SMALL LETTER BETA	voiced bilabial fricative
θ	U+03B8	GREEK SMALL LETTER THETA	voiceless dental fricative
χ	U+03C7	GREEK SMALL LETTER CHI	voiceless uvular fricative

Table 3.2: Strict-IPA diacritics with Unicode encodings

	Code	Unicode name	IPA name
◌̴	U+0334	COMBINING TILDE OVERLAY	velarized or pharyngealized
◌̼	U+033C	COMBINING SEAGULL BELOW	linguolabial
◌̪	U+032A	COMBINING BRIDGE BELOW	dental
◌̻	U+033B	COMBINING SQUARE BELOW	laminal
◌̺	U+033A	COMBINING INVERTED BRIDGE BELOW	apical
◌̟	U+031F	COMBINING PLUS SIGN BELOW	advanced
◌̠	U+0320	COMBINING MINUS SIGN BELOW	retracted
◌̝	U+031D	COMBINING UP TACK BELOW	raised
◌̞	U+031E	COMBINING DOWN TACK BELOW	lowered
◌̘	U+0318	COMBINING LEFT TACK BELOW	advanced tongue root
◌̙	U+0319	COMBINING RIGHT TACK BELOW	retracted tongue root

continued on next page

Table 3.2 Strict-IPA diacritics with Unicode encodings — *continued*

	Code	Unicode name	IPA name
◌̜	U+031C	COMBINING LEFT HALF RING BELOW	less rounded
◌̹	U+0339	COMBINING RIGHT HALF RING BELOW	more rounded
◌̬	U+032C	COMBINING CARON BELOW	voiced
◌̥	U+0325	COMBINING RING BELOW	voiceless
◌̰	U+0330	COMBINING TILDE BELOW	creaky voiced
◌̤	U+0324	COMBINING DIAERESIS BELOW	breathy voiced
◌̩	U+0329	COMBINING VERTICAL LINE BELOW	syllabic
◌̯	U+032F	COMBINING INVERTED BREVE BELOW	non-syllabic
◌̃	U+0303	COMBINING TILDE	nasalized
◌̈	U+0308	COMBINING DIAERESIS	centralized
◌̽	U+033D	COMBINING X ABOVE	mid-centralized
◌̆	U+0306	COMBINING BREVE	extra-short
◌˞	U+031A	COMBINING LEFT ANGLE ABOVE	no audible release
◌ʶ	U+02DE	MODIFIER LETTER RHOTIC HOOK	rhotacized
◌ˡ	U+02E1	MODIFIER LETTER SMALL L	lateral release
◌ⁿ	U+207F	SUPERSCRIPT LATIN SMALL LETTER N	nasal release
◌ʷ	U+02B7	MODIFIER LETTER SMALL W	labialized
◌ʲ	U+02B2	MODIFIER LETTER SMALL J	palatalized
◌ˠ	U+02E0	MODIFIER LETTER SMALL GAMMA	velarized
◌ˤ	U+02E4	MODIFIER LETTER SMALL REVERSED GLOTTAL STOP	pharyngealized
◌ʰ	U+02B0	MODIFIER LETTER SMALL H	aspirated
◌ʼ	U+02BC	MODIFIER LETTER APOSTROPHE	ejective
◌ː	U+02D0	MODIFIER LETTER TRIANGULAR COLON	long
◌ˑ	U+02D1	MODIFIER LETTER HALF TRIANGULAR COLON	half-long
◌͡◌	U+0361	COMBINING DOUBLE INVERTED BREVE	tie bar

Table 3.3: Other Strict-IPA symbols with Unicode encodings

Code	Unicode name	IPA name

continued on next page

Table 3.3 Other Strict-IPA symbols with Unicode encodings — *continued*

	Code	Unicode name	IPA name
ˈ	U+02C8	MODIFIER LETTER VERTICAL LINE	primary stress
ˌ	U+02CC	MODIFIER LETTER LOW VERTICAL LINE	secondary stress
˥	U+02E5	MODIFIER LETTER EXTRA-HIGH TONE BAR	extra high tone
˦	U+02E6	MODIFIER LETTER HIGH TONE BAR	high tone
˧	U+02E7	MODIFIER LETTER MID TONE BAR	mid tone
˨	U+02E8	MODIFIER LETTER LOW TONE BAR	low tone
˩	U+02E9	MODIFIER LETTER EXTRA-LOW TONE BAR	extra low tone
↑	U+2191	UPWARDS ARROW	global rise
↓	U+2193	DOWNWARDS ARROW	global fall
↗	U+2197	NORTH EAST ARROW	global rise
↘	U+2198	SOUTH EAST ARROW	global fall
	U+0020	SPACE	word break
.	U+002E	FULL STOP	syllable break
|	U+007C	VERTICAL LINE	minor group break (foot)
‖	U+2016	DOUBLE VERTICAL LINE	major group break (intonation)
‿	U+203F	UNDERTIE	linking (absence of a break)

Table 3.4: Additional characters for valid-IPA with Unicode encodings

	Code	Unicode name	Phonetic description
s̥	U+030A	COMBINING RING ABOVE	voiceless (above)
g	U+0067	LATIN SMALL LETTER G	voiced velar plosive
a̋	U+030B	COMBINING DOUBLE ACUTE ACCENT	extra high tone
á	U+0301	COMBINING ACUTE ACCENT	high tone
ā	U+0304	COMBINING MACRON	mid tone
à	U+0300	COMBINING GRAVE ACCENT	low tone
ȁ	U+030F	COMBINING DOUBLE GRAVE ACCENT	extra low tone
â	U+0302	COMBINING CIRCUMFLEX ACCENT	falling
ǎ	U+030C	COMBINING CARON	rising
a᷅	U+1DC4	COMBINING MACRON-ACUTE	high rising

continued on next page

Table 3.4 Additional characters for valid-IPA with Unicode encodings — *continued*

	Code	Unicode name	Phonetic description
◌᷅	U+1DC5	COMBINING GRAVE-MACRON	low rising
◌᷄	U+1DC6	COMBINING MACRON-GRAVE	low falling
◌᷆	U+1DC7	COMBINING ACUTE-MACRON	high falling
◌᷇	U+1DC8	COMBINING GRAVE-ACUTE-GRAVE	rising-falling
◌᷈	U+1DC9	COMBINING ACUTE-GRAVE-ACUTE	falling-rising
◌͡◌	U+035C	COMBINING DOUBLE BREVE BELOW	tie bar (below)

Table 3.5: Additions to widened-IPA with Unicode encodings

	Code	Unicode name	Phonetic description
‼	U+203C	DOUBLE EXCLAMATION MARK	retroflex click
ɗ	U+1D91	LATIN SMALL LETTER D WITH HOOK AND TAIL	voiced retroflex implosive
◌͈	U+0348	COMBINING DOUBLE VERTICAL LINE BELOW	fortis
◌͉	U+0349	COMBINING LEFT ANGLE BELOW	lenis
◌͓	U+0353	COMBINING X BELOW	frictionalized
◌͚	U+032E	COMBINING BREVE BELOW	derhoticized
◌͇	U+0347	COMBINING EQUALS SIGN BELOW	non-sibilant
◌ˀ	U+02C0	MODIFIER LETTER GLOTTAL STOP	glottalized
ɦ◌	U+02B1	MODIFIER LETTER SMALL H WITH HOOK	voiced pre-aspirated
◌ᴱ	U+1D31	MODIFIER LETTER CAPITAL E	epilaryngeal phonation

4 Orthography profiles

4.1 Characterizing writing systems

At this point in the course of rapid ongoing developments, we are left with a situation in which the Unicode Standard offers a highly detailed and flexible approach to deal computationally with writing systems, but it has unfortunately not influenced the linguistic practice very much. In many practical situations, the Unicode Standard is far too complex for the day-to-day practice in linguistics because it does not offer practical solutions for the down-to-earth problems of many linguists. In this section, we propose some simple practical guidelines and methods to improve on this situation.

Our central aims for linguistics, to be approached with a Unicode-based solution, are: (i) to improve the consistency of the encoding of sources, (ii) to transparently document knowledge about the writing system (including transliteration), and (iii) to do all of that in a way that is easy and quick to manage for many different sources with many different writing systems. The central concept in our proposal is the ORTHOGRAPHY PROFILE, a simple tab-separated CSV text file, that characterizes and documents a writing system. We also offer basic implementations in Python and R to assist with the production of such files, and to apply orthography profiles for consistency testing, grapheme tokenization and transliteration. Not only can orthography profiles be helpful in the daily practice of linguistics, they also succinctly document the orthographic details of a specific source, and, as such, might fruitfully be published alongside sources (e.g. in digital archives). Also, in high-level linguistic analyzes in which the graphemic detail is of central importance (e.g. phonotactic or comparative-historical studies), orthography profiles can transparently document the decisions that have been taken in the interpretation of the orthography in the sources used.

Given these goals, Unicode locale descriptions (see Section **??**) might seem like the ideal orthography profiles. However, there are various practical obstacles preventing the use of such locale descriptions in the daily linguistic practice, namely: (i) the XML-structure is too verbose to easily and quickly produce or correct manually, (ii) locale descriptions are designed for a wide scope on information (like

date formats or names of weekdays) most of which is not applicable for documenting writing systems, and (iii) most crucially, even if someone made the effort to produce a technically correct locale description for a specific source at hand, then it is nigh impossible to deploy the description. This is because a locale description has to be submitted to and accepted by the Unicode Common Locale Data Repository. The repository is (rightly so) not interested in descriptions that only apply to a limited set of sources (e.g. only a single specific dictionary).

The major challenge then is developing an infrastructure to identify the elements that are individual graphemes in a source, specifically for the enormous variety of sources using some kind of alphabetic writing system. Authors of source documents (e.g. dictionaries, wordlists, corpora) use a variety of writing systems that range from their own idiosyncratic transcriptions to already well-established practical or longstanding orthographies. Although the IPA is one practical choice as a sound-based normalization for writing systems (which can act as an interlingual pivot to attain interoperability across writing systems), graphemes in each writing system must also be identified and standardized if interoperability across different sources is to be achieved. In most cases, this amounts to more than simply mapping a grapheme to an IPA segment because graphemes must first be identified in context (e.g. is the sequence one sound or two sounds or both?) and strings must be tokenized, which may include taking orthographic rules into account (e.g. between vowels is /n/ and after a vowel but before a consonant is a nasalized vowel /ṽ/). In our experience, data from each source must be individually tokenized into graphemes so that its orthographic structure is identified and its contents can be extracted. To extract data for analysis, a source-by-source approach is required before an orthography profile can be created. For example, almost each available lexicon on the world's languages is idiosyncratic in its orthography and thus requires lexicon-specific approaches to identify graphemes in the writing system and to map graphemes to phonemes, if desired.

Thus, our key proposal for the characterization of a writing system is to use a grapheme tokenization as an inter-orthographic pivot. Basically, any source document is tokenized by graphemes, and only then a mapping to IPA (or any other orthographic conversion) is performed. An orthography profile then is a description of the units and rules that are needed to adequately model a graphemic tokenization for a language variety as described in a particular source document. An orthography profile summarizes the Unicode (tailored) graphemes and orthographic rules used to write a language (the details of the structure and assumptions of such a profile will be presented in the next section).

As an example of graphemic tokenization, note the three different levels of technological and linguistic elements that interact in the hypothetical lexical form <tsʰõ̰shi>:

1. code points (10 text elements): t s ʰ o ̰ ̃ ́ s h i
2. grapheme clusters (7 text elements): t s ʰ õ̰ s h i
3. tailored grapheme clusters (4 text elements): tsʰ õ̰ sh i

In (1), the string <tsʰõ̰shi> has been tokenized into ten Unicode code points (using NFD normalization), delimited here by space. Unicode tokenization is required because sequences of code points can differ in their visual and logical orders. For example, <õ̰> is ambiguous to whether it is the sequence of + ◌̰ + ◌̃> or + ◌̃ + ◌̰>. Although these two variants are visually homoglyphs, computationally they are different. Unicode normalization should be applied to this string to reorder the code points into a canonical order, allowing the data to be treated canonically equivalently for search and comparison. In (2), the Unicode code points have been logically normalized and visually organized into grapheme clusters, as specified by the Unicode Standard. The combining character sequence <õ̰> is normalized and visually grouped together. Note that, the MODIFIER LETTER SMALL H at U+02B0, is not grouped with. This is because it belongs to Spacing Modifier Letters category in the Unicode Standard. These characters are underspecified for the direction in which they modify a host character. For example, can indicate either pre- or post-aspiration (whereas the nasalization or creaky diacritic is defined in the Unicode Standard to apply to a specified base character). Finally, to arrive at the graphemic tokenization in (3), tailored grapheme clusters are needed (as for example specified in an orthography profile). For example, this orthography profile would specify that the sequence of characters, and form a single grapheme, and that and form a grapheme. The orthography profile could also specify orthographic rules, e.g. when tokenization graphemes, in say English, the in the forms and should be treated as distinct sequences depending on their contexts.

4.2 Informal description

An orthography profile describes the Unicode code points, characters, graphemes and orthographic rules in a writing system. An orthography profile is a language-specific (and often even resource-specific) description of the units and rules that are needed to adequately model a writing system. An important assumption of our work is that we assume a resource is encoded in Unicode (or

has been converted to Unicode). Any data source that the Unicode Standard is unable to capture, will also not be captured by an orthography profile.

Informally, an orthography profile specifies the graphemes (or, in Unicode parlance, TAILORED GRAPHEME CLUSTERS) that are expected to occur in any data to be analyzed or checked for consistency. These graphemes are first identified throughout the whole data (a step which we call TOKENIZATION), and possibly simply returned as such, possibly including error messages about any parts of the data that are not specified by the orthography profile. Once the graphemes are identified, they might also be changed into other graphemes (a step which we call TRANSLITERATION). When a grapheme has different possible transliterations, then these differences should be separated by contextual specification, possibly down to listing individual exceptional cases.

In practice, we foresee a workflow in which orthography profiles are iteratively refined, while at the same time inconsistencies and errors in the data to be tokenized are corrected. In some more complex use-cases there might even be a need for multiple different orthography profiles to be applied in sequence (see Section 5 on various exemplary use-cases). The result of any such workflow will normally be a cleaned dataset and an explicit description of the orthographic structure in the form of an orthography profile. Subsequently, the orthography profiles can be easily distributed in scholarly channels alongside the cleaned data, for example in supplementary material added to journal papers or in electronic archives.

4.3 Formal specification

The formal specifications of an orthography profile (or simply PROFILE for short) are the following:

1. A PROFILE IS A UNICODE UTF-8 ENCODED TEXT FILE (ideally using NFC, no-BOM, and LF; see Section 2.11, Pitfall: File Formats) that includes the information pertinent to the orthography.
2. A PROFILE IS A TAB-SEPARATED CSV FILE WITH AN OBLIGATORY HEADER LINE. A minimal profile can have just a single column, in which case there will of course no tabs, but the first line will still be the header. For all columns we assume the name in the header of the CSV file to be crucial. The actual ordering of the columns is unimportant.
3. LINES STARTING WITH A HASH <#> ARE IGNORED. Comments and metadata can be included inside the file, but only as complete lines in the profile, to

be marked by lines starting with hash # (NUMBER SIGN at U+0023). Hashes somewhere else in the file are to be treated literally, i.e. hashes are only to be ignored when they occur at the start of a line.[1]

4. METADATA ARE GIVEN IN COMMENTED LINES AT THE BEGINNING OF THE TEXT FILE IN A BASIC TAG: VALUE FORMAT. Metadata about the orthographic description given in the orthography profile includes, minimally, (i) author, (ii) date, (iii) title, (iv) a stable language identifier encoded in BCP 47/ISO 639-3, and (v) bibliographic data for resource(s) that illustrate the orthography described in the profile.

The content of a profile consists of lines, each describing a grapheme of the orthography, using the following columns:

1. A MINIMAL PROFILE CONSISTS OF A SINGLE COLUMN WITH HEADER GRAPHEMES, listing each of the different graphemes in a separate line.
2. OPTIONAL COLUMNS CALLED LEFT AND RIGHT CAN BE USED TO SPECIFY THE LEFT AND RIGHT CONTEXT OF THE GRAPHEME, RESPECTIVELY. The same grapheme can occur multiple times with different contextual specifications, for example to distinguish different pronunciations depending on the context.
3. THE COLUMNS GRAPHEME, LEFT AND RIGHT CAN USE REGULAR EXPRESSION METACHARACTERS. If regular expressions are used, then all literal usage of the special symbols, like full stops <.> or dollar signs <$> (so-called METACHARACTERS) have to be explicitly escaped by adding a backslash before them (i.e. use <.> or <$>). Note that any specification of context automatically expects regular expressions, so it is probably better to always escape all regular expression metacharacters when used literally in the orthography, i.e. the following symbols will need to be preceded by a backslash: [ ] ( ) { } + * . - ! ? ^ $.
4. AN OPTIONAL COLUMN CALLED CLASS CAN BE USED TO SPECIFY CLASSES OF GRAPHEMES, for example to define a class of vowels. Users can simply add ad-hoc identifiers in this column to indicate a group of graphemes, which can then be used in the description of the graphemes or the context. The identifiers should of course be chosen such that they do not conflate with

[1] Comments that belong to specific lines will have to be put in a separate column of the CSV file, e.g. add a column called COMMENTS. Further, if the content of a profile contains a hash at the start of a line, either reorder the columns so the hash does not occur at the start of the line, or add a dummy column in front of the data to not have the data start with a hash.

any symbols used in the orthography themselves. Note that such classes only refer to the graphemes, not to the context.

5. Columns describing transliterations for each graphemes can be added and named at will. Often more than a single possible transliteration will be of interest. Any software application using these profiles should use the names of these columns to select a specific transliteration column.
6. Any other columns can be added freely, but will mostly be ignored by any software application using the profiles. As orthography profiles are also intended to be read and interpreted by humans, it is often highly useful to add extra information on the graphemes in further columns, like for example Unicode codepoints, Unicode names, frequency of occurrence, examples of occurrence, explanation of the contextual restrictions, or comments.

For the automatic processing of the profiles, the following technical standards will be expected:

1. Each line of a profile will be interpreted as a regular expression. Software applications using profiles can also offer to interpret a profile in the literal sense to avoid the necessity for the user to escape regular expressions metacharacters in the profile. However, this only is possible when no contexts or classes are described, so this seems only useful in the most basic orthographies.
2. The class column will be used to produce explicit or chains of regular expressions, which will then be inserted in the graphemes, left and right columns at the position indicated by the class-identifiers. For example, a class V as a context specification might be replaced by a regular expression like: (a|e|i|o|u|ei|au). Only the graphemes themselves are included here, not any contexts specified for the elements of the class.
3. The left and right contexts will be included into the regular expressions by using lookbehind and lookahead. Basically, the actual regular expression syntax of lookbehind and lookahead is simply hidden to the users by allowing them to only specify the contexts themselves. Internally, the contexts in the columns left and right are combined with the column graphemes to form a complex regular expression like: (?<=left)graphemes(?=rig
4. The regular expressions will be applied in the order as specified in the profile, from top to bottom. A software implementation can offer

help in figuring out the optimal ordering of the regular expressions, but should then explicitly report on the order used.

The actual implementation of the profile on some text-string will function as follows:

1. ALL GRAPHEMES ARE MATCHED IN THE TEXT BEFORE THEY ARE TOKENIZED OR TRANSLITERATED. In this way, there is no necessity for the user to consider 'feeding' and 'bleeding' situations, in which the application of a rule either changes the text so another rule suddenly applies (feeding) or prevents another rule to apply ('bleeding').
2. THE MATCHING OF THE GRAPHEMES CAN OCCUR EITHER GLOBALLY OR LINEARLY. From a computer science perspective, the most natural way to match graphemes from a profile in some text is by walking linearly through the text-string from left to right, and at each position go through all graphemes in the profile to see which one matches, then go to the position at the end of the matched grapheme and start over. This is basically how a finite state transducer works, which is a well-established technique in computer science. However, from a linguistic point of view, our experience is that most linguists find it more natural to think from a global perspective. In this approach, the first grapheme in the profile is matched everywhere in the text-string first, before moving to the next grapheme in the profile. Theoretically, these approaches will lead to different results, though in practice of actual natural language orthographies they almost always lead to the same result. Still, we suggest that any software application using orthography profiles should offer both approaches (i.e. GLOBAL or LINEAR) to the user. The approach used should be documented in the metadata as TOKENIZATION METHOD.
3. THE MATCHING OF THE GRAPHEMES CAN OCCUR EITHER IN NFC OR NFD. By default, both the profile and the text-string to be tokenized should be treated as NFC (see section 2.9, Pitfall: Canonical equivalence, above). However, in some use-cases it turns out to be practical to treat both text and profile as NFD. This typically happens when very many different combinations of diacritics occur in the data. An NFD-profile can then be used to first check which individual diacritics are used, before turning to the more cumbersome inspection of all combinations. We suggest that any software application using orthography profiles should offer both approaches (i.e. NFC or NFD) to the user. The approach used should be documented in the metadata as UNICODE NORMALIZATION.

4. The text-string is always returned in tokenized form by separating the matched graphemes by a user-specified symbols-string. Any transliteration will be returned on top of the tokenization.
5. Leftover characters (i.e. characters that are not matched by the profile) should be reported to the user as errors. Typically, the unmatched character are replaced in the tokenization by a user-specified symbol-string.

Any software application offering to use orthography profile:

1. should offer user-options to specify:
    1. the name of the column to be used for transliteration (if any).
    2. the symbol-string to be inserted between graphemes. Optionally, a warning might be given if the chosen string includes characters from the orthography itself.
    3. the symbol-string to be inserted for unmatched strings in the tokenized and transliterated output.
    4. the tokenization method, i.e. whether the tokenization should proceed global or linear.
    5. unicode normalization, i.e. whether the text-string and profile should use NFC or NFD.
2. might offer user-options to:
    6. assist in the ordering of the graphemes. In our experience, it makes sense to apply larger graphemes before shorter graphemes, and to apply graphemes with context before graphemes without context. Further, frequently relevant rules might be applied after rarely relevant rules (though frequency is difficult to establish in practice, as it depends on the available data). Also, if this all fails to give any decisive ordering between rules, it seems useful to offer linguists the option to reverse the ordering from any manual specified ordering, because linguists tend to write the more general rule first, before turning to exceptions or special cases.
    7. assist in dealing with upper and lower case characters. It seems practical to offer some basic case matching, so characters like <a> and <A> are treated equally. This will be useful in many concrete cases, although the user should be warned that case matching does not function universally in the same way across orthographies. Ideally, users should prepare orthography profiles with all lowercase

and uppercase variants explicitly mentioned, so by default no case matching should be performed.

8. TREAT THE PROFILE LITERAL, i.e. to not interpret regular expression metacharacters. Matching graphemes literally often leads to strong speed increase, and would allow users to not needing to worry about escaping metacharacters. However, in our experience all actually interesting use-cases of orthography profiles include some contexts, which automatically prevents any literal interpretation, so by default the matching should not be literal.

3. SHOULD RETURN THE FOLLOWING INFORMATION to the user:

9. THE ORIGINAL TEXT-STRINGS TO BE PROCESSED IN THE USED UNICODE NORMALIZATION, i.e. in either NFC or NFD as specified by the user.
10. THE TOKENIZED STRINGS, with additionally any transliterated strings, if transliteration is requested.
11. A SURVEY OF ALL ERRORS ENCOUNTERED, ideally both in which text-strings any errors occurred and which characters in the text-strings lead to errors.
12. A REORDERED PROFILE, when any automatic reordering is offered

4.4 Examples

[Here should a few abstract short simple examples be added]

Note that to deal with ambiguous parsing cases, we can use the Unicode approach using the zero width joiner. This is actually a non-joiner (the name is confusing): the idea is to add this character into the text to identify cases in which a sequence of characters is not supposed to be a complex grapheme (even though the sequence is in the orthography profile)

5 Use cases

5.1 Introduction

We now present several use cases that have motivated the development of orthography profiles. These include:

- tokenization and error checking
- normalization of orthographic systems to attain interoperability across different source documents,
- cognate identification for detecting the similarity between words from different languages
- ...

An important assumption of our work is that input sources are encoded in Unicode (UTF-8 to be precise). Anything that the Unicode Standard is unable to capture, cannot be captured by an orthography profile. We also use Unicode normalization form NFD to decompose all incoming text input into normalized and (Unicode) logically ordered strings. This process is of fundamental importance when working with text data in the Unicode Standard.

5.2 Tokenization and error checking

The most basic use case is tokenization of language data. Tokenization comes in three types:

- Unicode character tokenization
- grapheme tokenization
- tailored grapheme tokenization

The simplest form of tokenization provided by any Unicode Standard-compliant software will split an input text stream on Unicode character code points. The input text is split on the character bit sequences as they have been encoded by the user. This may be in various Unicode Normalization Forms (see Section ??)

and tokenization split function returns a byte stream sequence given a particular encoding (see Section **??**).

There is increasing support for the regular expression match, commonly “ X”, to identify Unicode graphemes (see Section **??**) and perform tokenization on these sets of Unicode characters. Examples are given in Section **??**.

The orthography profile provides the mechanism for tokenizing an input stream on tailored grapheme clusters. The orthography’s formal specification is given in Section **??**. To sum, an orthography profile is a list of tailored grapheme clusters and / or orthographic rules that specify how a specific resource of text input should be tokenized.

Once text has been normalized, an additional and straightforward process is Unicode grapheme tokenization. More recently this regular expression, often available as short cut “\X”, identifies all sequences of “base” characters followed by 0 or more combining diacritics. For example, the sequence, a :>. For a first pass at identifying grapheme clusters, this is a straightforward tokenization. However, the linguist will note that both the tie bar (COMBINING DOUBLE INVERTED BREVE, a Unicode Combining Modifier) and the length marker (MODIFIER LETTER TRIANGULAR COLON. a Unicode Letter Modifier) do not appear “correctly”, i.e. the tie bar is grouped with the first character in the sequence and the length marker appears singly. This is necessary, as defined by the Unicode Standard, because of these character’s semantic properties – with Unicode we cannot know if, for example, the MODIFIER LETTER TRIANGULAR COLON should appear before or after the base character that it modifies, cf. the IPA aspiration marker, , the MODIFIER LETTER SMALL H, which is also a Unicode Letter Modifier and for linguists a diacritic that be be used for pre- or post-aspiration. These ambiguities of position in tokenization are not uncommon in IPA, thus orthographic (or source) normalization and tokenization is needed.

5.3 Cross-orthographic analysis of writing systems

Given that orthography profiles are stored in a standard CSV format, we can use tools for converting and working with CSV. One such tool is the command line utility CSVKIT.[1]

[1] https://csvkit.readthedocs.org

5.4 PHOIBLE

As new languages' phonological inventories are added to the database,[2] updates and additions to the conventions are published online by **MoranMcCloy2014**[3] , like introducing several symbols for widely attested sounds that lack representation in the current version of the IPA. Whenever possible, additions were drawn from the extIPA symbols for disordered speech.[4]

MoranMcCloy2014 explicitly define all character sequences so that the vast variety of phonemes found in a large sample of descriptions of the world's language were normalized into consistent character sequences, e.g. if one language description uses <kʲ'> and another <k'ʲ>, when both are intended to be phonetically equivalent, then a decision to normalize to one form was taken.

Character sequences with diacritics above the base character were not problematic in Moran (2012) because they include only the centralized, mid-centralized and nasalized combining characters.

Moran2012;MoranMcCloy2014 mark tones as singletons with Space Modifier Letters, e.g. <˥> for a phonemic high tone, instead of accent diacritics, alleviating potential conflicts.

Moran2012;MoranMcCloy2014 represent clicks as a combination of anterior and posterior articulations, where the set < k q g ɢ ŋ ɴ > denote place of articulation and phonation type (voiced, voiceless, nasal). The anterior articulation click characters are < ʘ ǀ ǁ ǃ ǂ > with the addition of non-standard IPA < ‼ > DOUBLE EXCLAMATION MARK) U+203C to represent the retroflex click. Note that the (post)alveolar click < ǃ > at U+01C3 is called incorrectly LATIN LETTER RETROFLEX CLICK (it is also a different character than the keyboard < ! > EXCLAMATION MARK at U+0021). All sources that contain clicks in PHOIBLE are standardized to contain both the anterior and posterior characters and diacritics are added to the posterior glyph, e.g. < n̊ǃ >.[5]

Building on these works...

[2] http://phoible.org/
[3] http://phoible.github.io/conventions/
[4] https://www.internationalphoneticassociation.org/sites/default/files/extIPAChart2008.pdf
[5] http://phoible.github.io/conventions/

Bibliography

Anderson, Lloyd B. 1984. Multilingual text processing in a two-byte code. In *Proceedings of the 10th international conference on computational linguistics* (COLING '84), 1–4. Stanford, California: Association for Computational Linguistics. http://dx.doi.org/10.3115/980431.980492.

Association, International Phonetic et al. 1989. Report on the 1989 kiel convention. *Journal of the International Phonetic Association* 19(2). 67–80.

Becker, Joseph D. 1984. Multilingual word processing. *Scientific American* 251(1). 96–107.

Beider, Alexander & Stephen P. Morse. 2008. Beider-morse phonetic matching: an alternative to soundex with fewer false hits. *Avotaynu: the International Review of Jewish Genealogy* 24(2). 12. https://web.archive.org/web/20150513075152/http://stevemorse.org/phonetics/bmpm.htm.

Belongie, Serge, Jitendra Malik & Jan Puzicha. 2002. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(4). 509–522.

Bird, Steven & Gary F. Simons. 2003. Seven Dimensions of Portability for Language Documentation and Description. *Language* 79(3). 557–582. http://www.language-archives.org/documents/portability.pdf.

Brown, Cecil H., Eric W. Holman & Søren Wichmann. 2013. Sound correspondences in the world's languages. *Language* 89(1). 4–29.

Computer, Apple. 1985. The font manager. *Inside Machintosh* 1. 215–240.

Computer, Apple. 1986. The font manager. *Inside Machintosh* 4. 27–46.

Computer, Apple. 1988. The script manager. *Inside Machintosh* 5. 293–322.

Daniels, Peter T. 1990. Fundamentals of Grammatology. *Journal of the American Oriental Society* 110(4). 727–731.

Daniels, Peter T. 1996. The Study of Writing Systems. In P. T. Daniels & W. Bright (eds.), *The world's writing systems*. New York, NY: Oxford University Press.

Daniels, Peter T. & William Bright. 1996. *The World's Writing Systems*. New York, NY: Oxford University Press.

Dolgopolsky, A. B. 1986. A probabilistic hypothesis concerning the oldest relationships among the language families of northern eurasia. In Vitalij V. Shevoroshkin

(ed. and trans.). Trans. from the Russian by Vitalij V. Shevoroshkin, *Typology, relationship and time: A collection of papers on language change and relationship by soviet linguists*, 27–50. Ann Arbor: Karoma Publisher.

Esling, John. 1990. Computer coding of the ipa: supplementary report. *Journal of the International Phonetic Association* 20(01). 22–26.

Esling, John H. & Harry Gaylord. 1993. Computer codes for phonetic symbols. *Journal of the International Phonetic Association* 23(2). 83–97.

Evans, Nicholas & Stephen C Levinson. 2009. The myth of language universals: language diversity and its importance for cognitive science. *Behavioral and brain sciences* 32(05). 429–448.

Gaultney, J. Victor. 2002. *Problems of diacritic design for latin script text faces*. SIL International.

Hieronymus, James L. 1993. Ascii phonetic symbols for the world's languages: worldbet. *Journal of the International Phonetic Association* 23.

Huurdeman, Anton A. 2003. *The worldwide history of telecommunications*. New York, NY: John Wiley & Sons.

International Phonetic Association. 2005. *International Phonetic Alphabet*. Tech. rep. International Phonetic Association. http://www.arts.gla.ac.uk/IPA/.

Kemp, Alan. 2006. Phonetic transcription: history. *The Encyclopedia of Language and Linguistics* 6. 396–410.

Knuth, Donald Ervin. 1973. *The art of programming: sorting and searching*. Vol. 3. Reading, Mass.: Addison-Wesley.

Kohrt, Manfred. 1986. The Term 'Grapheme' in the History and Theory of Linguistics. In Gerhard Augst (ed.), *New trends in graphemics and orthography*, 80–96. Berlin: de Gruyter.

Ladefoged, Peter. 1990. Some Reflections on the IPA. *Journal of Phonetics* 18. 335–346.

List, Johann-Mattis. 2012a. Multiple sequence alignment in historical linguistics: A sound class based approach. In *Proceedings of console xix*, 241–260.

List, Johann-Mattis. 2012b. SCA. phonetic alignment based on sound classes. In Marija Slavkovik & Dan Lassiter (eds.), *New directions in logic, language, and computation* (LNCS 7415), 32–51. Berlin & Heidelberg: Springer.

List, Johann-Mattis. 2012c. SCA: phonetic alignment based on sound classes. In Daniel Lassiter & Marija Slavkovik (eds.), *New directions in logic, language, and computation*, 32–51. Berlin & Heidelberg: Springer.

List, Johann-Mattis & Steven Moran. 2013. An open source toolkit for quantitative historical linguistics. In *Proceedings of the 51st annual meeting of the association for computational linguistics*, 13–18.

Maddieson, Ian. 1984. *Pattern of Sounds*. Cambridge, UK: Cambridge University Press.

Mania, Hubert. 2008. *Gauss: eine biographie*. Reinbek bei Hamburg: Rowohlt.

Mc Laughlin, Fiona. 2005. Voiceless Implosives in Seereer-Siin. *Journal of the International Phonetic Association* 35(2). 201–214.

Meinhof, Carl & Daniel Jones. 1928. Principles of Practical Orthography for African Languages. *Africa: Journal of the International African Institute* 1(2). 228–239.

Meyer, Julien. 2015. *Whistled languages: a worldwide inquiry on human whistled speech*. Berlin: Springer. http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=957780.

Meyer, Julien, Laure Dentel & Frank Seifart. 2012. A methodology for the study of rhythm in drummed forms of languages: application to bora manguaré of amazon. In *Proceedings of interspeech*, 686–690.

Mielke, Jeff. 2009. Segment Inventories. *Language and Linguistics Compass* 3/2. 700–718.

Moran, Steven. 2006. *A grammatical sketch of western sisaala*. Ypsilanit, MI: Eastern Michigan University MA thesis.

Moran, Steven. 2012. *Phonetics Information Base and Lexicon*. University of Washington PhD thesis.

Moran, Steven & Jelena Prokić. 2013. Investigating the relatedness of the endangered dogon languages. *Literary And Linguistic Computing* 28(4). 676–691. http://llc.oxfordjournals.org/content/28/4/676.

Olúmúyìw, Tèmítópẹ́. 2013. Yoruba writing: standards and trends. *Journal of Arts and Humanities* 2(1). 40–51. http://www.theartsjournal.org/index.php/site/article/view/50.

Postel, Hans Joachim. 1969. Die kölner phonetik: ein verfahren zur identifizierung von personennamen auf der grundlage der gestaltanalyse. *IBM-Nachrichten* 19. 925–931.

Powell, Barry B. 2012. *Writing theory and history of the technology of civilization*. New York, NY: John Wiley & Sons. http://nbn-resolving.de/urn:nbn:de:101:1-201502112957.

Pullum, Geoffrey K.. & William Allen Ladusaw. 1986. *Phonetic symbol guide*. University of Chicago Press.

Robinson, Andrew. 1995. *The story of writing*. London: Thames & Hudson.

Simons, Gary F. 1989. Working With Special Characters. In *Occasional publications in academic computing* (Occasional Publications in Academic Computing). Dallas, TX: Summer Institute of Linguistics.

Simons, Gary F. 1996. The nature of linguistic data and the requirements of a computing environment for linguistic research. *Dutch Studies on Near Eastern Languages and Literature* 2(1). 111–128.

Singh, Simon. 1999. *The code book*. New York: Doubleday.

Sproat, Richard. 2000. *A Computational Theory of Writing Systems*. Cambridge, UK: Cambridge University Press.

Steiner, Lydia, Peter F Stadler & Michael Cysouw. 2011. A pipeline for computational historical linguistics. *Language Dynamics and Change* 1(1). 89–127.

The International Phonetic Association. 1999. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press.

The International Phonetic Association. 2007. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Vol. Eighth printing. Cambridge University Press.

Unicode Consortium, The. 2014. *The Unicode Standard, Version 7.0.0*. Tech. rep. Mountain View, CA: The Unicode Consortium.

Wells, John, William Barry, Martine Grice, Adrian Fourcin & Dafydd Gibbon. 1992. Standard computer-compatible transcription. *Esprit project 2589 (SAM), Doc. no. SAM-UCL* 37.

Wells, John C. 1987. Computer-coded phonetic transcription. *Journal of the International Phonetic Association* 17(02). 94–114.

Wells, John C. 1995. Computer-coding the ipa: a proposed extension of sampa. *Revised draft* 4(28). 1995.