



Communication Langagière

Ingénierie des langues et de la parole

1. Introduction générale
2. Ingénierie des langues
 - 2.1. Représentation et codage des textes
 - 2.2. Applications du TALN :
 - 2.2.1. Dictionnaire et étiquetage de surface
 - 2.2.2. Traduction automatique statistique
 - 2.3. Introduction à l'apprentissage profond
 - 2.4. Encodeur/Décodeur
 - 2.5. BERT
3. Ingénierie de la parole
 - 3.1. Rappels de traitement numérique du signal
 - 3.2. Codage et représentation de la parole
 - 3.3. Approches auto-supervisées



Représentation et codage des textes : Introduction à Unicode

- Transparents inspirés du cours d'introduction de S. Bortzmeyer
- Unicode
- représenter la quasi-totalité des écritures utilisées dans le monde
- conçu selon un modèle en couches : séparation du répertoire de caractères et de leur représentation en bits
- => différence entre « jeu de caractères » et « encodage »



Citation

•If you've never explored this stuff before, you may be wondering how foreign-language alphabets and keyboards worked before Unicode—when character codes were just 8 bits wide. Well, in short, it was a mess.

Charles Petzold





Unicode : quelques remarques préliminaires

- Langue et écriture
 - Deux choses différentes
 - Ex : turc passé de l'alphabet arabe à l'alphabet latin au début du 20^e siècle (pareil pour vietnamien romanisé au 18^e)
 - Certaines langues s'écrivent couramment avec deux alphabets comme le serbo-croate, qui utilise l'alphabet latin en Croatie et l'alphabet cyrillique en Serbie
- Unicode ne s'occupe que des écritures (pas des langues)
- Attention au concept de caractère
 - Écritures alphabétiques vs non-alphabétiques (idéogrammes)
- Variété du monde
 - Écritures gauche-droite, droite-gauche, en lacets,...
- Unicode gère des caractères abstraits, pas des formes



Modèle en couches

- Unicode n'est *pas* une police de caractères
- Unicode gère des caractères abstraits, pas des formes (glyphes)
- La norme Unicode présente des glyphes, mais à des fins d'illustration uniquement
- distinction entre, par exemple, le jeu de caractères et la représentation physique
- On va présenter les couches, en partant de la plus haute (la plus éloignée de la machine)



Modèle en couches

Jeu de caractères abstraits (<i>Abstract Character Repertoire</i>)
Jeu de caractères codés (<i>Coded Character Set</i>)
Forme codée en mémoire (<i>Character Encoding Form</i>) ou encodage
Mécanisme de sérialisation des caractères (<i>Character Encoding Scheme</i>)
Surcodage de transfert (<i>Transfer Encoding Syntax</i>) (optionnel)



Modèle en couches

Jeu de caractères abstraits (<i>Abstract Character Repertoire</i>)
Jeu de caractères codés (<i>Coded Character Set</i>)
Forme codée en mémoire (<i>Character Encoding Form</i>) ou encodage
Mécanisme de sérialisation des caractères (<i>Character Encoding Scheme</i>)
Surcodage de transfert (<i>Transfer Encoding Syntax</i>) (optionnel)



Jeu de caractères abstraits (*Abstract Character Repertoire*)

- La couche la plus élevée est la définition du jeu de caractères.
- Unicode normalise actuellement près de 100 000 caractères et leur donne des noms
 - voir norme ISO 10646(F) pour les noms officiels français
- Exemple :

Lettre majuscule latine a	A
Lettre majuscule latine c cédille	Ç
Lettre minuscule grecque lambda	λ



Modèle en couches

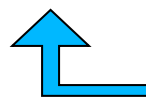
Jeu de caractères abstraits (<i>Abstract Character Repertoire</i>)
Jeu de caractères codés (<i>Coded Character Set</i>)
Forme codée en mémoire (<i>Character Encoding Form</i>) ou encodage
Mécanisme de sérialisation des caractères (<i>Character Encoding Scheme</i>)
Surcodage de transfert (<i>Transfer Encoding Syntax</i>) (optionnel)



Jeu de caractères codés (*Coded Character Set*)

- on ajoute à la table précédente un index numérique
- !! il ne s'agit pas d'une représentation en mémoire, juste d'un nombre
- Exemple :

U+0041	Lettre majuscule latine a	A
U+00C7	Lettre majuscule latine c cédille	Ç
U+03BB	Lettre minuscule grecque lambda	λ



Point de code
(code point)
val. hex.



Modèle en couches

Jeu de caractères abstraits (<i>Abstract Character Repertoire</i>)
Jeu de caractères codés (<i>Coded Character Set</i>)
Forme codée en mémoire (<i>Character Encoding Form</i>) ou encodage
Mécanisme de sérialisation des caractères (<i>Character Encoding Scheme</i>)
Surcodage de transfert (<i>Transfer Encoding Syntax</i>) (optionnel)




Forme codée en mémoire (*Character Encoding Form*) ou encodage

- spécifie quelles unités de stockage (*code units*) : octets ou mots de 16 ou de 32 bits, vont représenter un point de code
- particularité d'Unicode : il existe plusieurs encodages possibles
- Attention à la confusion entre Unicode et tel ou tel encodage



Modèle en couches

Jeu de caractères abstraits (<i>Abstract Character Repertoire</i>)
Jeu de caractères codés (<i>Coded Character Set</i>)
Forme codée en mémoire (<i>Character Encoding Form</i>) ou encodage
Mécanisme de sérialisation des caractères (<i>Character Encoding Scheme</i>)
Surcodage de transfert (<i>Transfer Encoding Syntax</i>) (optionnel)



Mécanisme de sérialisation des caractères (*Character Encoding Scheme*)

- sérialisation des unités de stockage
- Spécification big/little endian
- Définit le format des données vues par un observateur qui regarde ce qui passe sur le câble



Modèle en couches

Jeu de caractères abstraits (<i>Abstract Character Repertoire</i>)
Jeu de caractères codés (<i>Coded Character Set</i>)
Forme codée en mémoire (<i>Character Encoding Form</i>) ou encodage
Mécanisme de sérialisation des caractères (<i>Character Encoding Scheme</i>)
Surcodage de transfert (<i>Transfer Encoding Syntax</i>) (optionnel)



Surcodage de transfert (*Transfer Encoding Syntax*)

- Optionnel
- Intervention éventuelle de mécanismes de compression ou de chiffrement



Différents jeux de caractères

- **US-ASCII**

- ne comprend que l'alphabet latin, sans caractères composés (127 caractères en tout).
- caractères encodés sur 7 bits (inclus dans un octet dont le bit de poids fort est à zéro).
- Il ne convient qu'aux anglophones et aux communications entre ordinateurs

- **ASCII accentué**

- Ajout de caractères accentués (bit de poids fort à 1)
- ISO 646



Différents jeux de caractères

- **ISO-8859-X**

- désigne un ensemble de normes ISO
- en France ISO-8859-1, dite Latin-1 (**man iso_8859_1**)
- 255 caractères, encodage sur 8 bits (un octet)
- Remplace US-ASCII et "ASCII accentué"
- Latin-1 ne permet pas d'encoder tous les caractères utilisés en français! (ex : la ligature œ n'existe pas du tout en Latin-1)
- Latin-1 sera petit à petit remplacé par Latin-15, quasi-identique à l'exception du caractère € pour l'euro.
- Pour l'écriture grecque, on utilise ISO-8859-7. La partie haute (bit de poids fort à 1) sert pour l'alphabet grec et la partie basse pour l'alphabet latin
- Possibilité de programmer en C en mettant des commentaires en grec ou d'écrire un texte qui mêle le grec et l'anglais.



Pourquoi Unicode ?

- seul à permettre des textes multi-écritures (français et grec, par exemple)
- seul qui permet de traiter toutes les écritures (certaines écritures "rares" n'ont pas de jeu de caractères en dehors d'Unicode)
- Exemple : éditeur qui affichent les caractères Unicode (cf TP)

Les index

- Les 128 premiers caractères d'Unicode ont un index qui correspond aux caractères d'US-ASCII, pour des raisons pratiques évidentes
 - Ex : "d" a pour *code point* U+0064 comme en US-ASCII.
- Les 128 suivants ont le même index qu'en Latin-1
 - Ex : U+00FE pour le þ, le *thorn* scandinave.
- Rappelons qu'il ne s'agit que d'une identité des index, pas des représentations, un fichier Latin-1 ne sera pas un fichier Unicode légal



Les combinaisons

- Unicode gère les combinaisons de caractères.
 - exemple le ç (c cédille) peut être vu comme la combinaison de la lettre c et de la cédille combinatoire.
- **!!** Pour cette raison, pas d'équivalence entre les caractères au sens Unicode du terme et les caractères perçus par l'utilisateur !!
 - Si un fichier Unicode contient les deux points de code U+0063 (c) et U+0327 (cédille combinatoire), l'utilisateur qui l'affiche ne verra probablement qu'un seul caractère, ç.
- Pour des raisons de compatibilité, certaines combinaisons sont précomposées dans Unicode. le seul U+00E7 correspond lui aussi à ç
- => il ne faut ***jamais*** comparer deux chaînes de caractères Unicode sans leur avoir fait subir une normalisation



Les encodages

- Unicode permet plusieurs encodages
- Chaque encodage a des propriétés distinctes :
 - Place occupée en mémoire (8,16,32 bits)
 - Compatibilité avec les anciennes applications
 - Performances pour des opérations comme la sélection du Nième caractère (ex: un encodage où tous les caractères sont représentés avec la même taille est avantage).
- **UCS-2**
 - 16 bits (ne permet donc pas de représenter tout Unicode)
 - 65 535 premiers caractères d'Unicode (= plan multilingue de base ou BMP pour *Basic Multilingual Plane*)
 - comportent presque toutes les écritures couramment utilisées, seuls les idéogrammes chinois manquant largement. UCS-2 n'est guère utilisé, en raison de cette limite.



Les encodages

- **UCS-4 ou UTF-32**

- 32 bits
- est le seul où l'index est identique à la représentation
- Permet de coder environ quatre milliards de caractères !!
- Encore peu utilisé

- **UTF-16**

- Comme UCS-2, encodage 16-bits
- mécanisme d'échappement : les seizelets d'indirection (*surrogates*) qui lui permet de représenter tout Unicode
- Java utilise cet encodage en interne (Python aussi!)



Les encodages

- **UTF-8, l'encodage de l'Internet ?**

- UTF-8 ([RFC2279]) est le plus répandu.

- Encodage par défaut en XML

- Le plus compatible avec les anciennes applications (ex : un caractère ASCII a la même représentation en UTF-8 et en ASCII)

- Par exemple, on peut utiliser UTF-8 avec les noyaux Unix existants, si on veut des noms de fichier en Unicode

- **!!** un caractère Unicode a un nombre d'octets variable en UTF-8 (cf cédille)

- Pb extraction du Nième caractère d'une chaîne

- Perl utilise UTF-8 comme représentation interne des chaînes Unicode.

Exemples

Text	As Unicode Codepoints	As Encoded Bytes (hex)
big	U+0062 U+0069 U+0067	UTF-16 BE: 00 62 00 69 00 67 UTF-8: 62 69 67 ISO-8859-1: 62 69 67 US-ASCII: 62 69 67
groß	U+0067 U+0072 U+006F U+00DF	UTF-16 BE: 00 67 00 72 00 6f 00 df UTF-8: 67 72 6f c3 9f ISO-8859-1: 67 72 6f df US-ASCII: <i>(none)</i>
□ □	U+304D U+3044	UTF-16 BE: 30 4d 30 44 UTF-8: e3 81 8d e3 81 84 ISO-8859-1: <i>(none)</i> US-ASCII: <i>(none)</i>

Voir ex sur <http://www.jmdoudoux.fr/java/dej/chap-encodage.htm>



Unicode sur le Web

- Comment le navigateur sait quel encodage est utilisé ?
 - L'encodage peut être précisé dans le fichier
- HTML : `<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">`
- XML : `<?xml version="1.0" encoding="ISO-8859-1"?>`
 - L'utilisateur du navigateur peut fixer l'encodage
- Internet Explorer : menu View → Encoding
 - Le navigateur essaie de « deviner »
- Celui qui écrit la page peut spécifier explicitement l'utilisation de caractères Unicode
 - Ex : `&0x00A3` ou `£`



Unicode et Java

- <http://www.jmdoudoux.fr/java/dej/chap-encodage.htm>



Unicode et Python

- <https://docs.python.org/3/howto/unicode.html>



Problèmes de normalisation

- Si on compare deux chaînes de caractère Unicode en comparant leurs octets, on est sûr de se tromper
 - Elles n'utilisent pas forcément le même encodage et la comparaison physique des bits n'a pas de sens.
- => convertir tout en entrée, de façon à n'utiliser qu'un seul encodage en interne
- Il reste d'autres problèmes :
 - les caractères combinants (cédille)
 - certains caractères, quoique différents pour la norme Unicode (*code point* différents) sont équivalents dans un contexte donné (ex: *straße* et *strasse*)
 - Le problème est encore plus épineux en chinois où il existe plusieurs représentations du même caractère, depuis une réforme de la graphie.
- => Si on veut comparer deux chaînes de caractères Unicode, il faut donc normaliser les chaînes avant comparaison (RFC3454)



Autres problèmes

•Casse

- système de nommage indépendant de la casse facile en ASCII (on ajoute ou on retire 32 à l'encodage)
- pas en Unicode, où les tables sont bien plus complexes
- Unicode fournit une table de correspondance de casse (*case folding*) qu'on peut utiliser dans ses programmes.
- dans certains cas, le changement de casse peut avoir un résultat qui dépend du profil (*locale*), par exemple de la langue de l'utilisateur.

•Tri

- contrairement à US-ASCII, impossible de trier sur la représentation binaire : il faut introduire plus de « sémantique » et consulter des tables.
- Exemple : en Suède et en Finlande, le Ä apparaît après le Z dans les tris, (annuaire du téléphone par ex.)



Prise en charge d'Unicode dans les programmes

- Au minimum, un programme doit pouvoir réaliser des entrées/sorties en Unicode. En entrée, il devrait accepter au moins les encodages UTF-8 et UTF-16, en sortie également.
- Entre les deux, il ne doit *pas* modifier les caractères qu'il ne comprend pas (aucun programme ne peut connaître la sémantique de tous les caractères Unicode).
- À un niveau de support plus élevé, il devrait pouvoir :
 - S'il fait des tris, ne pas trier selon la représentation binaire, mais selon l'algorithme de tri d'Unicode.
 - S'il permet des recherches avec des expressions rationnelles, utiliser les expressions Unicode.



Support d'Unicode dans vos programmes

- Unicode nous oblige à reconsidérer pas mal de présupposés. Par exemple, que doit renvoyer `strlen` ?
 - Le nombre de graphèmes (ce que l'utilisateur appellerait "caractères") ?
 - Le nombre de *code points* (ce que l'informaticien appellerait "caractères") ?
 - Le nombre d'octets (ce que ferait un programme naïf) ?
 - Ainsi, la chaîne "U+0063 U+0327" représente un seul graphème (ç), mais nécessite deux points de code (si on normalisait, on pourrait la réduire à un seul) et un nombre d'octets qui dépend de l'encodage.
- La distinction entre ces différents sens (qui sont tous valables, à leur façon), se retrouve dans la plupart des langages de programmation, qui permettent de différencier entre eux.



Exemple en perl

```
# "character semantics" vs. "byte semantics" en Perl
# Dans le premier cas, on travaille au niveau du jeu de
# caractères Unicode.
# Dans le deuxième, on ne connaît que des octets, sans
# leur
# signification.
print $ustring->length(), "\n"; # Affiche le nombre de code
points
$utf8 = $ustring->utf8;
print length($utf8), "\n"; # Affiche le nombre d'octets
```



Langages de programmation

- **Perl**

- bon support d'Unicode (> version 5.8)
- **man perlunicode**

- **Java**

- Java n'utilise qu'Unicode depuis le début.

- **C**

- Pas de vrai support Unicode
- il existe des types de données suffisamment grands pour stocker les caractères Unicode (`wchar_t`) mais sans sémantique associée
- entrée *Character Set Handling* de la documentation de la GNU libc contient des indications à ce sujet.



Outils Unicode

- Editeur : Yudit
 - <http://yudit.org/download/>
- outil de normalisation : charlint
 - <http://www.w3.org/International/Charlint/>
- Convertisseur entre jeux de caractères : recode
 - <http://www.gnu.org/directory/GNU/recode.html>
- Algo de tri d'unicode
 - <http://www.unicode.org/reports/tr10/>
- Commande Unix : iconv



Quelques Liens Unicode

<http://www.unicode.org/>

- <http://www.linux.org/docs/ldp/howto/Unicode-HOWTO.html>

- <http://www.w3.org/TR/charmod/>

- [RFC2277] H.T. Alvestrand. *RFC 2277: IETF Policy on Character Sets and Languages*. 1998.

- [RFC2279] F. Yergeau. *RFC 2279: UTF-8, a transformation format of ISO 10646*. 1998.

- [RFC3454] P. Hoffman. M. Blanchet. *RFC 3454: Preparation of Internationalized Strings ("stringprep")*. 2002.