Chapitre 5

Conseils de mise en œuvre

Il est possible de mettre en œuvre un sous-ensemble important du standard Unicode comme s'il s'agissait d'*ASCII étendu* nécessitant peu de modifications aux usages informatiques en cours. Cependant, le standard Unicode prévoit le traitement de langues et d'écritures aux comportements plus complexes que ceux du français. Qu'il s'agisse d'implanter un nouveau système d'exploitation à partir de zéro ou d'améliorer un environnement applicatif ou de programmation, il faut étudier les conventions et façons de faire actuelles en informatique afin de les adapter à ces comportements plus complexes.

Ce chapitre aborde une série de brefs conseils utiles pour les développeurs responsables de la mise en œuvre d'Unicode. L'information et les exemples ci-joints visent à mieux faire comprendre la conception et les fonctionnalités du standard Unicode. Ces conseils sont donc destinés à promouvoir des normes professionnelles dans la mise en œuvre du standard Unicode.

Ces lignes directrices ne sont pas normatives et les développeurs ne sont donc pas obligés de s'y conformer.

5.1 Transcodage vers d'autres normes et standards

Le standard Unicode se présente dans un monde où coexiste déjà de nombreuses normes de codage de caractères privées, nationales, voire internationales. Un des principaux atouts d'Unicode est d'incorporer un grand nombre de normes et de standards. Le standard Unicode a souvent inclus des caractères en double afin de garantir un transcodage aller-retour vers les normes préétablies les plus utilisées.

Il n'est pas toujours aisé de convertir des caractères entre deux normes de codage. La signification de nombreux caractères est souvent plurielle et peut correspondre à celle de plusieurs caractères dans un autre jeu de caractères. Il arrive qu'une norme code en double un même caractère, parfois l'interprétation d'un ensemble complet de caractères dépend de l'application qui les utilise. Enfin, il existe de subtiles différences dans ce que différentes normes considèrent comme étant des caractères.

Problématique

Le standard Unicode peut s'utiliser comme une norme charnière qui permet de passer d'un jeu de caractères à l'autre. Ce processus, parfois appelé *transcodage par pivot* ou *triangularisation*, réduit le nombre de tableaux de transcodages nécessaires de $O(n^2)$ à $O(n)$, quand n est le nombre de jeux de caractères. En règle générale, on emploie des tableaux – plutôt que des conversions purement algorithmes – pour passer d'Unicode à un autre jeu de caractères. La consultation d'un tableau est habituellement bien plus rapide que les algorithmes de conversion, même les plus simples comme ceux qui permettent de passer de JIS à Shift-JIS.

Tableaux multiétapes

Les tableaux prennent de la place. Même les petits jeux convertissent leurs caractères vers des caractères appartenant à différents blocs du standard Unicode et peuvent donc nécessiter

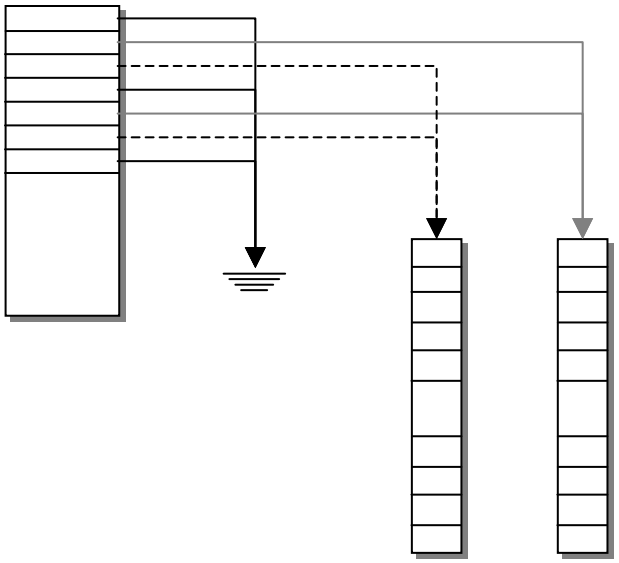
64k cellules[1] dans au moins une direction. Différentes techniques permettent de réduire les besoins en mémoire de ces tableaux. Ces techniques ne sont pas uniquement valables pour les tableaux de transcodage mais également pour de nombreux autres tableaux nécessaires à la mise en œuvre du standard Unicode : les propriétés des caractères, les tableaux de tri lexicographique, et les tableaux de sélection de glyphe.

Tableaux plats. Si on ne se soucie pas de l'espace mémoire, la taille et l'ensemble des pages de travail des systèmes d'exploitation à mémoire virtuelle sont suffisants même si on utilise des tableaux plats, car la fréquence d'utilisation des caractères varient énormément et même les petits jeux de caractères contiennent de nombreux caractères rarement utilisés. En outre, les données à transformer vers un jeu de caractères particulier ne proviennent en règle générale que de quelques blocs du standard Unicode. Ceci signifie que de grandes parties des tableaux de transcodage de 64k ne sont alors pas utilisées (celles qui contiennent le caractère par défaut ou les caractères qui ne sont pas transcodés) et que les pages correspondantes ne sont donc jamais chargés en mémoire.

Intervalles. On peut être tenté d'« optimiser » l'utilisation mémoire de ces tableaux en prenant en compte des intervalles imbriqués ou d'autres mécanismes similaires. Sur des systèmes modernes dont les instructions s'exécutent en parallèle dans un pipeline, la nature conditionnelle de cette technique la pénalise considérablement. Une méthode plus rapide consiste à utiliser un *tableau à deux étapes optimisé* que l'on peut programmer sans aucune instruction de test ni de saut. Les tableaux de hachage épargnent de la mémoire, mais cette méthode est moins rapide que les tableaux multiétapes.

Tableaux à deux étapes. Les tableaux à deux étapes constituent un mécanisme fréquemment utilisé pour réduire la taille des tableaux. Ils utilisent un vecteur de 256 pointeurs (pour l'octet supérieur) et une valeur par défaut. Si le pointeur vaut `NULL`, on utilise la valeur par défaut. Sinon le pointeur renvoie à un bloc de 256 valeurs indexé grâce à l'octet inférieur.

Figure 5-1. Tableau à deux étapes



[1] Par plan, or la version 3.1 définit quatre plans : le plan multilingue de base (00_{16}), le plan multilingue complémentaire (01_{16}), le plan idéographique complémentaire (02_{16}) et le plan complémentaire spécialisé ($0E_{16}$).

Tableaux à deux étapes optimisés. Lorsque deux blocs sont identiques, leurs pointeurs renvoient simplement vers un seul et même bloc. Pour des tableaux de transcodages, ceci ne se produit habituellement que lorsque les blocs en question renvoient à la valeur « par défaut » ou à la valeur « non transcodable ». Plutôt que d'utiliser des pointeurs `NULL` et une valeur par défaut, on crée un bloc « partagé » de 256 valeurs par défaut. Toutes les entrées du premier niveau pour lesquelles il n'existe pas de valeur de transcodage renvoient à ce bloc. On évite de la sorte tout code conditionnel, ce mécanisme permet d'obtenir des temps d'accès qui se rapprochent du temps d'accès du tableau « plat » mais avec une grande économie de mémoire.

Transmission sur 7 ou 8 bits

Certains protocoles de transmission utilisent des codes de commande ASCII pour maîtriser le flux de données. Des logiciels tristement célèbres, parmi lesquels ceux de courrier sur UNIX, limitent les données à 7 bits ASCII. Dans ces cas-là, on enchasse (ou surcode) le texte Unicode avant de le transmettre. Il existe un certain nombre de protocoles pour ce faire, parmi lesquels `uuencode`, `BinHex` ou `SCSU`[2]. Ces protocoles peuvent intégrer une fonction de compression dans la passe de surcodage afin de réduire les coûts induits par la transmission.

Le format transformé UTF-8, décrit à la *Section 2.3, Modèle de caractères*, et à la *Section 3.8, Transformations*, peuvent s'utiliser pour transmettre des données codées en Unicode à travers un canal de transmission à 8 bits. Le format transformé UTF-7, définit par le RFC-2152, peut également s'utiliser avec MIME.

Tableaux de correspondance

Afin d'assister et d'orienter les développeurs, le standard Unicode fournit une série de tableaux de correspondance entre Unicode et différents jeux de caractères sur le cédérom qui accompagne cet ouvrage. Ces tableaux comprennent quelquefois des correspondances multiples. La fonction principale de ces tableaux est d'identifier les caractères de ces normes dans le contexte du standard Unicode. La conversion de données entre le standard Unicode et d'autres normes dépendra souvent de l'application ou du contexte en question. Beaucoup de fabricants publient des tableaux de correspondance pour leur propres jeux de caractères.

Avertissement

Dans la mesure du possible, le contenu de tous les tableaux de correspondances a été vérifié par le consortium Unicode. Toutefois, le consortium Unicode ne garantit pas que les tableaux imprimés dans cet ouvrage ou sur le disque optique qui l'accompagne sont corrects dans les moindres détails, il n'est donc pas responsables des erreurs qui auraient pu se glisser dans ces tableaux de correspondances ni pour les erreurs dans les logiciels qui mettraient en œuvre ces tableaux. Si une ambiguïté d'interprétation devait se présenter, il incombe aux développeurs de consulter les normes internationales ou les standards industriels appropriés.

5.2 wchar_t ANSI/ISO

Grâce au type de caractère large `wchar_t`, la version ANSI/ISO de C permet l'inclusion de caractères larges de largeur fixe. Toutefois, selon cette norme, la largeur précise dépend du

[2] Voir le rapport technique Unicode n°6, *A Standard Compression Scheme for Unicode*.

compilateur. Cependant un caractère de l'ensemble exécutable et portable de C doit correspondre à son caractère large par simple addition de zéros initiaux. Les caractères Unicode qui appartiennent à l'intervalle ASCII U+0020 à U+007E satisfont à cette exigence. Ainsi, si une mise en œuvre utilise donc l'ASCII pour coder l'ensemble exécutable et portable de C, l'utilisation du jeu de caractères Unicode comme type sous-jacent à `wchar_t`, sous la forme UTF-16 ou UTF-32, satisfait aux exigences.

La largeur de `wchar_t` dépend du compilateur et peut être aussi étroite que 8 bits. De ce fait, les programmes qui désirent être portables, quel que soit le compilateur C ou C++ utilisé, doivent s'abstenir d'utiliser `wchar_t` pour stocker du texte Unicode. L'objectif du type `wchar_t` est de permettre le stockage de caractères larges définis par le compilateur, ces caractères peuvent être Unicode pour certains compilateurs. Les développeurs qui désirent mettre en œuvre UTF-16 peuvent néanmoins définir une macro ou un `typedef` (par exemple, `CARUNICO`) qui correspondra à un `unsigned short` ou à `wchar_t` selon le compilateur et la plate-forme cibles. Les développeurs qui préfèrent mettre en œuvre UTF-32 peuvent avoir recours à une macro ou à un `typedef` qui pourra être compilé sous la forme d'un `unsigned int` ou d'un `wchar_t`, selon le compilateur et la plateforme cibles. Lorsqu'on est sûr que `wchar_t` correspond à une valeur de 16 bits[3], il est possible que de tels macros ou typedefs soient prédéfinis (par exemple, `WCHAR` dans le cas de l'API Win32).

Sur les systèmes où le type de caractère autochtone ou `wchar_t` est implanté comme une quantité de 32 bits, une mise en œuvre peut utiliser la forme UTF-32 pour représenter les caractères Unicode.

Le modèle du C ISO/ANSI présente une faiblesse dans l'hypothèse qu'il fait que les caractères peuvent toujours être traités isolément. Les mises en œuvre qui désirent dépasser le modèle du C ISO/ANSI trouveront sans doute utile de mélanger les largeurs au sein de leur API. Ainsi une application pourra-t-elle posséder un wchar_t sur 32 bits et traiter à la fois des formes UTF-8, UTF-16 et UTF-32. Une autre aura peut-être à sa disposition un wchar_t sur 16 bits et traitera alors les chaînes sous les formes UTF-8 ou UTF-16, mais elle pourra offrir une API différente qui permettra de traiter les caractères individuels en tant que UTF-32 ou deux seizets UTF-16.

5.3 Caractères inconnus ou manquants

Cette section explore sommairement la manière dont les utilisateurs peuvent traiter les caractères qui ne sont pas pris en charge ou qui, bien qu'ils le soient, ne peuvent être rendus de manière lisible.

Codes de caractères non affectés ou d'usage privé

Il existe deux classes de valeurs de code de caractère que même les mises en oeuvre « complètes » du standard Unicode n'interprètent pas nécessairement de manière correcte :

- les valeurs de code de caractères non affectées ;
- les valeurs de code de caractères de la zone à usage privé pour lesquelles il n'existe pas d'accord privé.

[3] Les dernières versions ISO de C prévoient des types `uint`*xx*`_t`, où *xx* précise la largeur en bits de ces entiers non signés. Il semble donc plus sage de définir les caractères en termes de `uint16_t` ou `uint32_t` (plus particulièrement pour la version 3.1). Ces types C peuvent être petits ou grands boutiens, il faut s'en souvenir avant de transmettre ces données vers d'autres machines.

Une mise en œuvre ne doit pas chercher à interpréter de telles valeurs de code. Parmi les manières dont on peut envisager l'affichage de ces valeurs de code inconnues : l'impression de la valeur de code sous la forme de chiffres hexadécimaux, l'affichage d'une boîte noire ou blanche ou simplement ne rien afficher. Une mise en œuvre ne doit en aucun cas supposer quoi que ce soit d'autre au sujet des propriétés de ces caractères, ni éliminer aveuglément ces caractères. Elle ne doit pas de manière inconsidérée les transformer en d'autres valeurs.

Dans la pratique, les applications sont confrontées à des points de codes non affectés ou à des caractères d'usage privé inconnus. Ceci peut se produire, par exemple, lorsqu'un programme traite du texte provenant d'un système dont la mise en œuvre d'Unicode est ultérieure à la sienne et qui contient de nouveaux caractères normalisés. Pour que ces applications fonctionnent correctement, il faut attribuer des propriétés implicites aux points de code non affectés puisque le bon fonctionnement de certains algorithmes nécessite que tous les caractères en aient. Une compatibilité maximale impose que ces propriétés implicites ne soient pas uniformes pour toutes les zones de caractères non attribués.

En règle générale, les points de code (ou numéros de caractère) qui n'appartiennent pas aux caractères pris en charge s'afficheront à l'aide d'un glyphe de repli, une boîte noire par exemple. Cependant, les caractères de formatage et de commande ne doivent pas s'afficher bien qu'ils puissent avoir un effet visuel sur les autres caractères. On ignore également ces caractères sauf en ce qui a trait à des processus précis et bien définis : le tri ignore ainsi l'ANTI-LIANT SANS CHASSE. Afin de garantir un plus degré niveau de compatibilité entre les différentes versions du standard, on a réservé les intervalles U+2600..U+206F, U+FFF0..U+FFFC et U+E0000..U+E0FFF aux caractères de formatage et de commande (Catégorie générale = Cf). Les processus de rendu et de traitement doivent donc ignorer les points de code non affectés appartenant à ces intervalles.

L'algorithme bidirectionnel Unicode (cf. *Section 3.12*) attribue une catégorie bidirectionnelle à tous les points de code non affectés conformément à la directionalité prévue à l'avenir pour ces caractères.

L'algorithme de coupure de lignes (Annexe normative d'Unicode n° 14, *Line Breaking Properties*), dans sa section *Définitions*, attribue la propriété « XX » à tous les points de codes non affectés.

Dans son calcul de la chasse des caractères pour le rendu extrême-oriental, l'annexe normative d'Unicode n°11 (*East Asian Width*) comprend également une section sur les caractères non affectés ou d'usage privé.

L'annexe normative d'Unicode n° 15 (*Formes de normalisation Unicode,* voir le *Chapitre 6* de ce présent ouvrage) attribue aux caractères non affectés la Classe canonique combinatoire = 0 et ne leur associe aucune décomposition.

Caractères interprétables mais non affichables

Il peut arriver qu'une mise en œuvre lise un caractère Unicode attribué mais qu'elle soit incapable de l'afficher car il lui manque une police appropriée ou une autre raison l'empêche de rendre le caractère correctement.

Il se peut malgré tout qu'une mise en œuvre puisse satisfaire certaines requêtes des utilisateurs en triant, par exemple, les données, en indiquant le système d'écriture de ces caractères ou en les affichant par un mécanisme de repli. Une mise en œuvre est libre d'illustrer de manière différente les caractères non affichables (mais attribués) et les valeurs de code non attribuées en utilisant pour les caractères attribués des glyphes qui indiquent leur famille (un caractère typique de l'écriture inscrit dans un carré, par exemple).

Caractères réaffectés

Certains caractères, principalement les hangûls, faisaient partie des versions d'Unicode antérieures à la version 2.0 mais ont été réaffectés à d'autres numéros (voir les tableaux de transcodages sur le cédérom). Dans la mesure du possible, ces valeurs doivent être acceptées et converties en leurs valeurs Unicode 3.1 correspondantes. Dans certains cas, il se peut qu'une application prévue pour la version 3.1 d'Unicode doive transmettre des numéros de caractères de la version 1.1 afin de pouvoir communiquer avec certaines applications prévues pour la version 1.1. Ce problème ne se présente que dans le cas de quelques anciennes applications et fichiers Unicode, il s'agit d'une complication peu fréquente. Le consortium Unicode a comme politique de dorénavant ne plus jamais réaffecter des caractères.

5.4 Traitement des paires d'indirection

Les paires de seizets d'indirection permettent de coder 917 476 caractères en UTF-16 sans avoir à recourir à des caractères de 32 bits. Ce mécanisme sert essentiellement à accéder des caractères rares, il n'est pas nécessaire que toutes les mises en œuvre sachent pour l'instant traiter ces paires. Unicode 3.1 définit près de 45.000 nouveaux caractères complémentaires[3] accessibles à l'aide de ces seizets.

Les valeurs des seizets d'indirection supérieurs et inférieurs appartiennent à des intervalles disjoints. Ces seizets ne permettent d'adresser que les caractères qui sont affectés aux plans complémentaires[4]. L'appartenance des seizets supérieurs et inférieurs à des intervalles disjoints signifie qu'il faut tout au plus lire l'unité de stockage précédente ou suivante, sans se soucier de tout autre contexte, pour déterminer la frontière d'un caractère. Cette méthode permet d'accéder efficacement toute position d'une chaîne UTF-16, ce qui n'est pas possible avec des codages comme Shift-JIS.

Dans un texte bien formé, un seizet inférieur d'indirection doit être immédiatement précédé par un seizet supérieur d'indirection. Il ne peut être précédé d'un autre seizet inférieur ou d'une unité de stockage qui n'est pas un seizet d'indirection.

Les seizets d'indirection ont également l'avantage d'être transparents pour les mises en œuvre qui ne les reconnaissent pas. Ainsi, la suite valable d'unités de stockage Unicode [0053] [0061] [006C] [0075] [0074] [0020] [D800] [DC00] [0021] [0021] sera interprétée par une mise en œuvre conforme à la version 1.0 comme « Salut<non reconnu><non reconnu>!! ». Ce résultat est à peine moins bon que celui produit par une mise en œuvre conforme à la version 3.0 qui ne prend pas en charge la paire de seizets d'indirection en question et qui interprètera cette suite comme « Salut<non reconnu>!! ».

Tant qu'une mise en œuvre n'élimine pas un des seizets d'indirection et qu'elle n'insère pas de caractère entre deux seizets d'indirection, elle préserve l'intégrité des données. Même si les données devaient être corrompues, la corruption demeurera localisée contrairement à certains codages multioctets comme Shift-JIS ou EUC. La corruption d'une unité de stockage Unicode n'altère qu'un seul caractère. Le fait que les seizets d'indirection supérieurs et inférieurs soient disjoints et qu'ils se présentent toujours en paire empêche les erreurs de se propager au reste du texte.

On peut classer les mises en œuvre des seizets d'indirection en fonctions de deux critères importants :

[4] Qui n'appartiennent pas au plan multilingue de base, lequel comprend les caractères U+0000..U+FFFF.

- La mise en œuvre interprète correctement la paire de seizets d'indirection.
- La mise en œuvre garantit l'intégrité de la paire de seizets d'indirection.

Ces deux options déterminent trois niveaux raisonnables de prise en charge des seizets d'indirection (voir le *Tableau 5-1*).

Tableau 5-1. Niveau de prise en charge des seizets d'indirection

Prise en charge	Interprétation	Intégrité des paires
Aucune	pas de paires	Non garantie
Faible	sous-ensemble de paires non nulles	Non garantie
Forte	sous-ensemble de paires non nulles	Garantie

Exemple. La phrase « La lettre grecque α correspond à <hiéroglyphe-supérieur> <hiéroglyphe-inférieur> et à <phénicien-supérieur><phénicien-inférieur> » peut être affichée de trois manières différentes, selon la présence ou non d'une police phénicienne et l'absence d'une police hiéroglyphique. Les ■ du *Tableau 5-2* représentent les caractères non interprété ables par la mise en œuvre.

Tableau 5-2. Exemples de niveaux de seizets d'indirection

Aucun	« La lettre grecque α correspond à■ ■ et à■■. »
Faible	« La lettre grecque α correspond à■ ■ et à <*phénicien*> . »
Fort	« La lettre grecque α correspond à■ et à <*phénicien*> . »

Il est facile d'adapter les mises en œuvre qui implantent des caractéristiques complexes du standard Unicode afin de mettre en œuvre le niveau inférieur de prise en charge des seizets d'indirection[5]. Par exemple :

- Le tri lexicographique des seizets d'indirection peut s'effectuer en considérant ces paires comme des « caractères groupés » similaires au « ij » néerlandais ou au « ll » du tri traditionnel castillan.
- La saisie des seizets d'indirection peut s'implanter en s'assurant qu'un caractère saisi engendre deux unités de stockage, de la même manière qu'un clavier arabe peut posséder une touche « *lam-alif* » qui engendre les deux caractères *lam* et *alif*.
- L'affichage et le calcul de la taille à l'écran des seizets d'indirection s'apparentent aux traitements des ligatures comme « f » et « i » qui se joignent pour former un seul glyphe « fi ».
- La troncature des seizets d'indirection repose sur le même mécanisme que celui utilisé pour s'assurer que les signes combinatoires accompagnent leur caractère de base. Pour plus d'informations, veuillez vous référer à la *Section 5.15*, *Repérage des frontières d'élément textuel.*

Pour empêcher les utilisateurs de corrompre le texte, les éditeurs de texte doivent s'assurer que les points d'insertion (également appelés *caret* ou barre d'insertion) correspondent aux frontières de caractère. À l'instar de la frontière d'élément textuel, il n'est pas nécessaire de modifier les routines de traitement de bas niveau des chaînes de caractères (comme `wcschr`) pour prévenir la corruption des paires de seizets d'indirection. En pratique, il suffit que certains processus de niveau supérieur (comme ceux mentionnés ci-dessus) gèrent les paires d'indirection ; les routines de niveau inférieur peuvent continuer à fonctionner sur des suites

[5] Depuis 3.0.1, il est évidemment possible de traiter les caractères comme des unités de stockage de 32 bits (UTF-32) ce qui simplifie sensiblement tous ces traitements.

d'unités de stockage Unicode de 16 bits sans aucun traitement spécial destinés aux seizets d'indirection.

5.5 Traitement des nombres

De nombreux groupes de caractères du standard Unicode représentent des chiffres décimaux appartenant à différentes écritures. Les systèmes qui interprètent ces caractères comme des nombres doivent fournir la valeur décimale correcte. Ainsi, la suite U+0968 २ CHIFFRE DÉVANÂGARÎ DEUX, U+0966 ० CHIFFRE DÉVANÂGARÎ ZÉRO doit équivaloir à vingt (décimal).

Lors de la conversion de valeurs numériques binaires, les chiffres utilisés peuvent provenir de différentes écritures. Ainsi, la valeur décimale *vingt* peut se représenter sous les formes d'un U+0032 2 CHIFFRE DEUX, U+0030 0 CHIFFRE ZÉRO, ou d'un U+0968 २ CHIFFRE DÉVANÂGARÎDEUX, U+0966 ० CHIFFRE DÉVANÂGARÎ ZÉRO ou encore d'un U+0662 ٢ CHIFFRE ARABE-HINDI DEUX, U+0660 ٠ CHIFFRE ARABE-HINDI ZÉRO. Le standard Unicode préconise que les systèmes permettent aux utilisateurs de choisir le format approprié des chiffres et de remplacer, par exemple, les occurrences de U+0030 0 CHIFFRE ZÉRO par des U+0660 ٠ CHIFFRE ARABE-HINDI ZÉRO. (Voir le *Chapitre 4*, *Propriétés des caractères*, et les tableaux nécessaires pour formater et analyser les valeurs numériques.)

Les variantes à pleine chasse des chiffres ASCII ne sont qu'une variante de compatibilité des chiffres ordinaires et doivent être considérées comme des chiffres européens ordinaires.

Les chiffres romains et les idéogrammes numéraux sont des systèmes d'écriture décimaux mais il ne s'agit pas, à strictement parler, de systèmes à base dix. Il est donc impossible de transposer bijectivement des chiffres comme 12456,789. Ces deux systèmes ne permettent d'écrire que des chiffres entiers positifs.

Notons également qu'il existe deux manières des chiffres à l'aide des idéogrammes numéraux. La *Figure 5-2* illustre les deux façons dont on peut écrire 1 234.

Figure 5-2. Idéogrammes numéraux

一千二百三十四

ou

一二三四

Afin de pouvoir analyser ces caractères numéraux, les mises en œuvre doivent être assez intelligentes pour pouvoir discerner ces deux cas.

On trouve souvent des chiffres qui ne font pas partie d'un nombre mais qu' il faut analyser. Les identificateurs alphanumériques constituent un de ces cas (v. la *Section 5.16*, *Identificateurs*). Ce n'est qu'à un second niveau (quand on met en œuvre par exemple, un parseur de formules mathématiques complet) que l'on doit considérer des questions cruciales comme l'interprétation des exposants.

5.6 Traitement des propriétés

Le standard Unicode fournit des informations détaillées au sujet des propriétés des caractères (v. le *Chapitre 4*, *Propriétés des caractères*, et la *Base de données des caractères Unicode* sur le disque optique qui accompagne ce livre). Les développeurs peuvent utiliser ces propriétés pour mettre en œuvre une litanie de processus de bas niveau. Des processus de haut niveau complètement adaptés au niveau linguistique réclament des données supplémentaires.

Il est possible d'utiliser un tableau à deux étapes, décrit à la *Section 5.1*, *Transcodage vers d'autres normes et standards*, pour accéder aux propriétés ou à d'autres informations d'un caractère. De la sorte, on peut représenter en mémoire de manière très efficace la *Base de données de caractères* présente sur le disque optique.

De nombreux caractères partagent souvent une même propriété, on peut donc stocker ces propriétés sous la forme de blocs partagés.

Beaucoup de mises en œuvre s'inspirent du modèle POSIX qui prévoient des fonctions séparées, comme `isalpha`, `isdigit` et ainsi de suite. Les développeurs de systèmes Unicode ou de bibliothèques d'internationalisation doivent prendre soin d'étendre correctement ces concepts à tous les caractères Unicode.

Les caractères combinatoires jouent un rôle de premier plan dans un texte codé à l'aide d'Unicode. Les développeurs et les concepteurs de langage informatique doivent avoir à l'esprit qu'il ne suffit pas d'indiquer quel caractère est combinatoire mais qu'un caractère combinatoire hérite les propriétés du caractère de base précédent (voir également les sections *3.5*, *Combinaison*, et *5.16*, *Identificateurs*). D'autres propriétés importantes, comme le poids lexicographique, peuvent également dépendre du contexte.

Le standard Unicode fournit un ensemble substantiel de propriétés. En conséquence, il peut être utile aux mises en œuvre d'accéder plusieurs propriétés à la fois et de renvoyer une chaîne de champs binaires, un champ binaire pour chaque caractère de la chaîne passée en argument.

Par le passé, de nombreuses normes comme le langage C supposait l'existence de « jeux de caractères portables » minimalistes dans la conception de fonctions opérant sur des caractères. Unicode devient, de plus en plus, *le* jeu de caractères portable. Lors de la mise en œuvre de processus textuels, il faut donc différencier les limitations historiques des véritables besoins actuels.

5.7 Normalisation

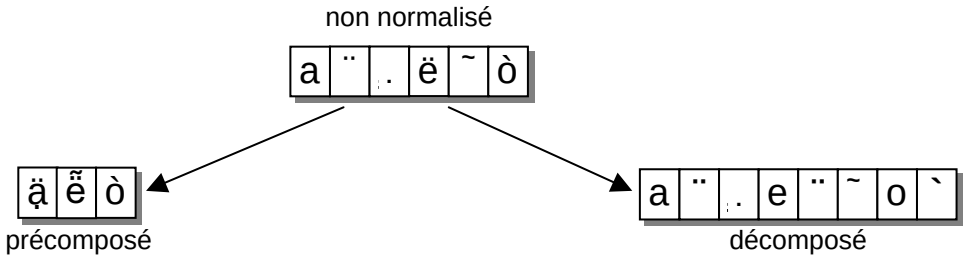
Variantes orthographiques. Le standard Unicode attribue un numéro de caractère aux lettres accentuées les plus fréquentes. Ces mêmes caractères s'obtiennent également par la composition d'un caractère de base et d'un ou plusieurs diacritiques.

Le standard Unicode fournit un tableau de variantes orthographiques (de décompositions) pour les caractères précomposés qui peuvent être composées à l'aide d'un caractère de base et d'un ou plusieurs signes à chasse nulle. Grâce à ces tableaux, on peut harmoniser le codage des chaînes conformément à la *Section 3.10*, *Mise en ordre canonique*. Les mises en œuvre « tolérantes » envers les données qu'elles reçoivent et « strictes » envers celles qu'elles émettent éprouveront le moins de problèmes de compatibilité.

Les tableaux de décomposition sont associés à une version particulière du standard Unicode. L'ajout ultérieur de nouveaux caractères pourrait occasionner des changements. L'ajout de nouveaux caractères précomposés s'accompagnera de l'ajout des formes décomposées correspondantes.

Normalisation. Les systèmes sont libres de normaliser un texte codé en Unicode sous une forme particulière. Ainsi, ils peuvent normaliser les suites de caractères en une suite de caractères précomposés ou vice-versa (voir la Figure 5-3).

Figure 5-3. Normalisation



Seul un petit nombre de formes accentuées précomposées potentielles se sont vu attribuer un numéro de caractère Unicode. La plupart de ces formes retenues proviennent de normes préexistantes.

Les systèmes qui ne peuvent pas gérer les signes à chasse nulle peuvent normaliser les textes Unicode sous la forme de caractères précomposées ; cette méthode permet de traiter la plupart des langues modernes utilisant l'écriture latine. Ces systèmes peuvent également indiquer visuellement, grâce à une méthode de repli, la valeur des combinaisons qu'ils ne peuvent traiter (cf. la sous-section *Rendu de repli* de la *Section 5.14*, *Rendu des signes à chasse nulle*).

Les systèmes qui peuvent gérer les signes à chasse nulle normalisent habituellement les textes Unicode afin d'éliminer les formes précomposées. Cette méthode permet d'adopter une représentation et un traitement homogènes des caractères accentués. Cependant, dans la plupart des cas, il n'est pas trop compliqué de traiter les formes composées et précomposées à la fois. C'est souvent la solution la plus simple.

Les formes standardisées de normalisation et les clauses de conformité à ces formes sont définies au *Chapitre 6*, *Formes de normalisation*. Pour d'autres renseignements, consultez le *Chapitre 3*, *Conformité*, le *Chapitre 4*, *Propriétés des caractères* et la *Section 2.6*, *Caractères combinatoires*.

5.8 Compression

L'utilisation du codage de caractères Unicode peut augmenter l'espace mémoire ou disque attribué aux éléments textuels. Il se peut donc qu'il soit souhaitable de comprimer les fichiers ou les chaînes codés en Unicode. La compression constitue toujours un protocole de niveau supérieur. Il faut donc connaître le mécanisme de compression pour que l'échange de données soit assuré. Pour un examen de la problématique liée à la compression et une présentation d'un mécanisme de compression standardisé pour Unicode, veuillez vous référer au rapport technique Unicode n° 6 *A Standard Compression Scheme for Unicode* présent sur le cédérom ou, pour une version tenue à jour, consulter le site Internet du Consortium Unicode.

Les formes de codage définies à la *Section 2.3*, *Modèles de caractères*, possèdent des caractéristiques de stockage différentes. Ainsi, tant qu'un texte ne contient que des caractères du bloc latin de base (ASCII), il occupe la même place qu'il soit codé en UTF-8 ou en ASCII. Par contre, un texte formé d'idéogrammes stocké à l'aide d'UTF-8 nécessitera plus d'espace que la forme stockée à l'aide d'UTF-16.

5.9 Traitement des lignes

Différentes plates-formes représentent souvent de manières différentes les passages à la ligne : CR, LF, NL et CRLF. Non seulement les passages à la ligne sont-ils représentés différemment sur des machines différentes, ils ont également des comportements ambigus sur une même plate-forme. Sur l'Internet où un texte sur une même machine peut provenir de plusieurs sources, cette incohérence occasionne de sérieux ennuis.

Malheureusement, ces caractères sont souvent transcodés directement dans leurs valeurs Unicode correspondantes. C'est pourquoi, même les programmes qui traitent du texte Unicode pur doivent se préoccuper de ces problèmes.

Pour des conseils détaillés à ce sujet, veuillez consulter le rapport technique Unicode n° 13 *Unicode Newline Guidelines* présent sur le cédérom ou, pour une version tenue à jour, consulter le site Internet du Consortium Unicode.

5.10 Expressions régulières

Il importe de modifier les moteurs à expressions régulières opérant sur des octets afin qu'ils traitent correctement Unicode. Les problèmes suivants doivent être considérés lors de cette modification :

- Unicode est un grand jeu de caractères – les moteurs à expressions régulières conçus pour de petits jeux de caractères pourraient mal supporter ce changement d'échelle.
- Unicode englobe de nombreux types d'écritures et langues qui possèdent des caractéristiques fort différentes du français ou des autres langues d'Europe occidentale.

Pour de plus amples informations, veuillez vous référer au rapport technique Unicode n° 18 *Unicode Regular Expression Guidelines* présent sur le cédérom ou, pour une version tenue à jour, consulter le site Internet du Consortium Unicode.

5.11 Renseignements linguistiques en texte brut

Nécessité des étiquettes linguistiques

On surestime souvent la nécessité d'incorporer des étiquettes linguistiques dans des données textuelles brutes. De nombreuses opérations habituelles comme le tri ont rarement besoin de cette information supplémentaire. Dans le cas d'un tri, les mots étrangers sont généralement triés comme s'il appartenait à la langue du document au complet. (Cf. le rapport technique Unicode n°10 *Unicode Collation Algorithm* présent sur le cédérom ou, pour une version tenue à jour, consulter le site Internet du consortium Unicode.) Ainsi, dans l'index d'un livre en français, le mot espagnol « churo » ne sera pas trié à l'espagnole après le mot « czar », pas plus que la ville suédoise d'Östersjö n'apparaîtra dans un atlas français après Zanzibar, alors que ce sera le cas dans un atlas suédois.

Toutefois, dans un document multilingue, des étiquettes linguistiques peuvent s'avérer très utiles pour certaines opérations comme la vérification orthographique ou la coupure de mots. Ces étiquettes permettent également de choisir la police par défaut d'un passage textuel non formaté. Les points de suspension, par exemple, sont rendus d'une manière différente selon que la police est japonaise ou européenne. Bien qu'une indication de la langue puisse être

utile lors de la synthèse vocale, les logiciels modernes capables de synthèse vocale incluent des fonctionnalités d'analyse grammaticale de texte tellement avancées que le travail supplémentaire nécessaire à la détermination de la langue est négligeable.

L'information linguistique peut être transmise sous la forme d'une étiquete insérée dans le texte ou de données parallèles au texte. Cette dernière méthode ne perturbe pas le traitement normal du texte (comparaison, recherche, etc.) et permet une manipulation plus aisée du texte.

Étiquettes linguistiques et l'unification han

Unification han. On croit souvent à tort que les caractères han unifiés ne peuvent être affichés correctement sans une indication de la langue utilisée. Cette idée préconçue peut faire croire aux développeurs qu'il faut toujours ajouter des étiquettes linguistiques aux textes bruts. Cette conclusion est néanmoins fausse. L'unification han a été conçue et mise en œuvre dans le but de conserver la lisibilité des textes. Bien qu'il faille spécifier une police, une taille, une chasse et d'autres informations de formatage pour reproduire exactement l'apparence du texte original, le texte brut reste lisible en l'absence de ces précisions.

Un texte han codé en Unicode n'est pas équivoque car tout caractère unifié appartient à l'intervalle des variantes styliques de ce caractère pour les différents styles d'idéogrammes unifiés. L'unification d'un caractère ne signifie pas qu'il devient illisible quand il est rendu avec une autre police. Quand de l'information de formatage est importante, le formatage d'un texte riche constitue sans doute la meilleure façon de la transmettre.

Scénarios typiques. Les scénarios d'échange de courriel suivants démontrent qu'on surestime souvent le besoin de balisage linguistique pour les textes hans.

- Scénario 1. Un utilisateur japonais envoie un texte japonais non étiqueté. Les lecteurs sont japonais (ont une police japonaise). Les lecteurs ne perçoivent pas de différences par rapport à ce qu'ils s'attendaient à voir.
- Scénario 2. Un utilisateur japonais envoie un texte sino-japonais non étiqueté. Les lecteurs sont japonais (ont une police japonaise) et chinois (ont une police chinoise). Les lecteurs lisent le texte mixte avec une seule police, néanmoins le texte est toujours lisible. Les lecteurs reconnaissent les différentes langues grâce au contexte.
- Scénario 3. Un utilisateur japonais envoie un texte sino-japonais. Le texte est étiqueté d'informations relatives à la police, le corps, la chasse,… car il importe de préserver au mieux le format. Les lecteurs ont les polices appropriées. Les lecteurs lisent le texte mixte, chaque langue avec sa police. Ils reconnaissent les différentes langues par leur contenu et lisent chaque langue dans une police plus typique à cette langue.

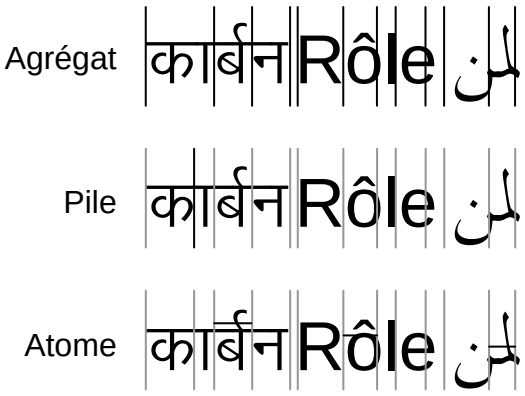
Pour des raisons de familiarité, il est fréquent, même dans les ouvrages de labeur, d'imprimer les passages en langue étrangère à l'aide de polices habituelles pour la langue de l'ouvrage.

5.12 Édition et sélection

Éléments contextuels cohérents

On obtient un éditeur de texte est plus intuitif quand les éléments textuels sont cohérents (cf. la *Figure 5-4*). Les opérations de suppression, de sélection, les clics de la souris et les mouvements du curseur doivent se comporter comme s'ils agissaient sur un ensemble cohérent de frontières. Ainsi, appuyer sur la touche flèche-arrière doit modifier la position du curseur de la même manière que la suppression d'un caractère. *Cette harmonisation fournit un modèle unique et cohérent d'édition des caractères.*

Figure 5-4. Frontières de caractères cohérentes.



Il existe trois types de frontières habituellement utiles dans le contexte de l'édition et de la sélection de mots.

Frontières d'agrégat. Les frontières d'agrégat se présentent dans des écritures comme la dévânagarî. La sélection ou la suppression d'un agrégat signifie que l'agrégat au complet (par exemple *ka* + *signe voyelle a*) ou le caractère composé (*o* + *circonflexe*) est sélectionné ou supprimé en un seul coup.

Frontières de pile. Les frontières de pile sont habituellement plus fines que les frontières d'agrégats. Les éléments autonomes (comme la *voyelle signe a*) peuvent alors être sélectionnés et supprimés indépendamment. Les éléments qui « s'empilent » (par exemple, *o* + *circonflexe* ou les ligatures verticales comme le *lam+mîm* arabe) constituent une unité de sélection. Les frontières de pile regroupent toutes les suites de caractères composés en unités distinctes, comme s'il s'agissait de caractères précomposés.

Frontières de caractère atomique. La méthode des frontières de caractère atomique est ce qui se rapproche le plus de la sélection des caractères Unicode isolés. Toutefois, comme la plupart des systèmes modernes indiquent la sélection à l'aide d'un rectangle de surlignage, cette méthode restreint la cohérence de l'édition car certaines suites de caractères ne se présentent pas en ordre linéaire. Quand les caractères s'empilent, il vaut mieux utiliser deux autres mécanismes de visualisation des caractères sélectionnés : les frontières linéaires et non linéaires.

Frontières linéaires. Pour les frontières linéaires, on considère que la chasse entière du glyphe résultant d'une suite de caractères appartient au premier caractère de cette suite et que les autres caractères ont une chasse nulle et qu'ils s'affichent par la suite. Cette solution est le mécanisme le plus simple ; c'est celui qu'utilise le système Macintosh et d'autres systèmes à

l'heure actuelle. L'avantage de cette méthode est qu'elle nécessite très peu de développement supplémentaire. Son désavantage : la difficulté de sélectionner les caractères étroits, pour ne pas parler des caractères à chasse nulle. Pour ce faire, l'utilisateur doit placer le curseur à la droite du signe sans chasse et glisser vers la gauche. Cette méthode ne permet pas de sélectionner individuellement les signes à chasse nulle si plus d'un est présent.

Frontières non linéaires. La méthode des frontières non linéaires divise tout élément empilé en morceaux. Ainsi, un point au milieu d'une ligature *lam* + *mîm* (ﻠ) peut-il représenter la division entre ces caractères. On peut ensuite permettre le surlignage de plusieurs rectangles ou choisir un autre mécanisme comme colorier chaque caractère différemment.

Remarquons qu'avec un peu plus de travail, un caractère précomposé peut se comporter lors de la suppression comme s'il s'agissait d'une suite de caractères composés à frontières de caractère atomique. Ce processus suppose que l'on décompose les caractères à la volée pour les utiliser dans une simulation. Simulation de la suppression qui implique décomposition, suppression du dernier caractère et recomposition (s'il reste plus d'un caractère). Cette technique ne fonctionne cependant pas en général pour l'édition et la sélection.

En effet, pour la plupart des systèmes, la plus petite unité textuelle adressable est le caractère. La sélection et l'assignation de propriétés (comme la police, la couleur, l'interlettrage,…) se fait au niveau du caractère. Il n'existe pas de manière satisfaisante de simuler cette adressabilité avec des caractères précomposés. La modification systématique de toutes les opérations d'édition pour qu'elles puissent se référer à toutes les fractions d'un caractère se révèlerait fort inefficace.

Tout comme il n'existe pas une définition unique de ce qui constitue un élément textuel, il n'existe pas une conception unique de ce qui constitue une frontière de caractères lors de l'édition. Il se peut même que l'utilisateur veuille avoir à sa disposition différents degrés de granularité d'édition. Deux méthodes viennent à l'esprit. La première permet à l'utilisateur de spécifier une préférence globale quant à la frontière des caractères. La seconde offre à l'utilisateur d'autres mécanismes de commande, comme la touche Majuscule-Supprimer, qui lui permet de régler de façon plus précise (ou imprécise) la valeur par défaut.

5.13 Stratégies pour traiter les signes à chasse nulle

Les lignes directrices suivantes devraient permettre de mettre en œuvre des systèmes et des fonctions qui prennent en charge de manière efficace les signes à chasse nulle. Le développeur est libre de choisir entre des techniques sommaires valables pour la grande majorité des systèmes actuels et des techniques plus complexes plus appropriées dans des situations plus exigeantes, comme les systèmes d'éditique (publication assistée par ordinateur) de pointe.

Dans cette section et la section suivante, les termes *signe à chasse nulle* et *caractère combinatoire* sont synonymes. Les termes *diacritique*, *accent*, *accent tonique*, *point hébreu*, *point-voyelle* et *voyelle arabe* s'utilisent parfois à la place de *signe à chasse nulle* (il s'agit d'instances de signe à chasse nulle).

La prise en charge des signes à chasse nulle nécessite un petit nombre de fonctionnalités. Plusieurs niveaux de prise en charge sont envisageables. Le système minimal fournit de bons résultats tout en étant relativement simple à mettre en œuvre : la plupart des fonctionnalités nécessaires ne sont que des améliorations au logiciel existant.

Les signes à chasse nulle sont indispensables pour la prise en charge d'écritures comme l'arabe, l'hébreu et les langues du sous-continent indien. De nombreux fabricants ont donc des systèmes qui prennent déjà en charge ces caractères à chasse nulle, on peut donc en profiter

pour développer des progiciels capables de traiter ces caractères « exotiques » dans le standard Unicode.

Rendu. Une substitution assez simple permet d'afficher efficacement un ensemble fixe de suites de caractères composés. Chaque fois qu'un caractère de base est suivi d'un ou plusieurs signes combinatoires sans chasse, on remplace ces caractères par un glyphe qui représentent la forme combinée de ces caractères. Dans le cas d'un affichage sommaire à chasse fixe, les caractères combinatoires à chasse nulle ne modifient pas la chasse du glyphe correspondant au caractère de base. Lors de la troncature d'une chaîne, il est toujours plus facile de tronquer en commençant par la fin et en remontant vers le début. De la sorte, un signe à chasse nulle ne se verra pas séparé du caractère de base qui le précède.

Un système de rendu plus perfectionné peut prendre en compte des variations plus subtiles au niveau de la chasse ou du crénage des signes à chasse nulle ainsi que les cas où la suite de caractères composés possède une chasse différente de celle du caractère de base. De tels systèmes de rendu ne sont pas nécessaires à la grande majorité des applications. Ils peuvent, néanmoins, également fournir des routines plus perfectionnées de troncature. (Voir également la *Section 5.14*, *Rendu des signes à chasse nulle*.)

Autres processus. Des routines de comparaison multilingue doivent être capables de comparer une suite de caractères comme s'il s'agissait d'un caractère et vice-versa. Ces routines peuvent donc aussi traiter les suites de caractères composés. Lors de recherches au sein d'une chaîne, il faut se souvenir de vérifier la présence de signes à chasse nulle supplémentaires qui peuvent influencer l'interprétation du dernier caractère comparé.

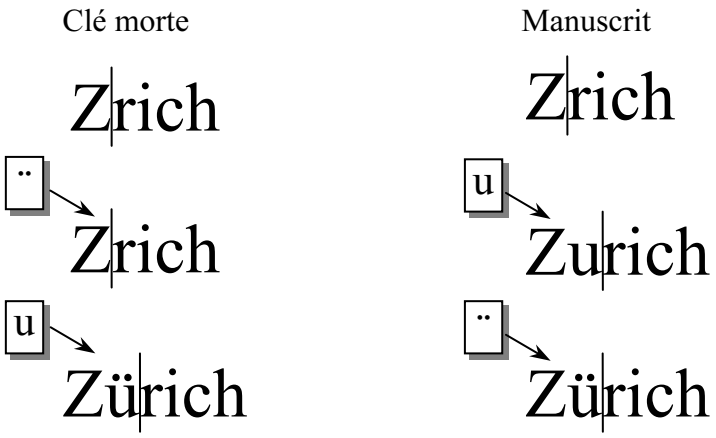
Les algorithmes de coupure de ligne utilisent en règle générale des automates à états pour calculer la fin des mots. On adapte aisément ces algorithmes pour qu'ils conservent les signes à chasse nulle avec leur caractère de base. (Voir la discussion aux sections *5.17*, *Tri et repérage*, *5.7*, *Normalisation* et *5.15*, *Repérage des frontières d'élément textuel.)*

Saisie au clavier

On met souvent en œuvre la saisie des suites de caractères composés à l'aide de *touches* dites *mortes*. Ces touches correspondent au mécanisme utilisé par les machines à écrire pour produire ces mêmes suites : la frappe successive au même endroit sur le papier du caractère de base et des accents. Les claviers informatiques passent à état particulier lorsque qu'on frappe une touche morte (pour ajouter par exemple un diacritique) et n'émettent un caractère précomposé que lorsqu'un des caractères de base « valables » suit. Il est aisé d'adapter ces systèmes pour qu'ils émettent des suites de caractères composés ou, au besoin, des caractères précomposés. Bien que les dactylographes, surtout ceux qui utilisent une écriture latine, sont habitués à d'abord taper les accents, la saisie de nombreuses écritures du standard Unicode (y compris l'écriture latine) peut s'effectuer dans l'ordre manuscrit où les accents s'écrivent après la lettre de base (voir la *Figure 5-5*).

Dans l'ordre manuscrit, chaque caractère saisi produit un caractère différent et évident, aucun état n'est caché. Pour adjoindre un accent à un caractère existant, il suffit à l'utilisateur de placer le point d'insertion (le « caret » ou barre d'insertion) après ce caractère avant de taper l'accent.

Figure 5-5. Différences entre suite à clés mortes et manuscrites



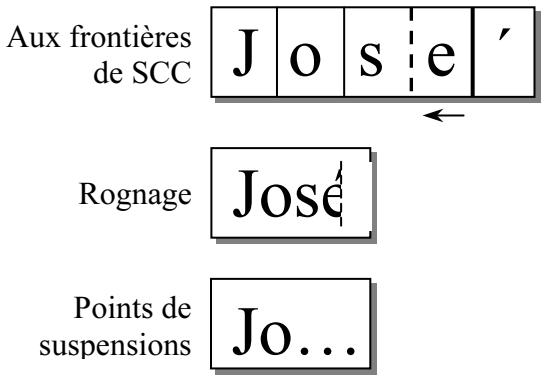
Troncature

Il existe deux genres de troncature : après un certain nombre de caractères et au-delà d'une certaine taille à l'affichage. La troncature en fonction du nombre de caractères peut occasionner une perte d'information.

On utilise la troncature selon le nombre de caractères quand, par manque de place, on ne peut stocker qu'un nombre limité de caractères. C'est le cas, par exemple, lorsqu'il faut diviser un texte en trames avant sa transmission. On évite alors les pertes en recomposant le texte en entier avant tout autre traitement ou en s'assurant qu'aucune suite de caractères combinatoire ne chevauche deux trames.

Lorsqu'on insère des données dans un champ de longueur fixe, il se peut que l'on perde de l'information. Il vaut souvent mieux tronquer à la frontière d'un élément textuel (par exemple, celle de la dernière suite de caractères composites ou même la frontière du dernier mot) plutôt que de le faire après la dernière unité de stockage (cf. la *Figure 5-6*). (Voir la *Section 5.15*, *Repérage des frontières d'élément textuel*).

Figure 5-6. Troncature des suites de caractères composés



La troncature en fonction de la taille occupée au rendu s'utilise lorsqu'on manque de place à l'écran ou sur le papier. On préfère utiliser alors la taille au rendu au nombre de caractères comme critère de troncature. Dans les systèmes rudimentaires, il est plus simple de tronquer selon la taille en commençant par la fin pour remonter vers le début du texte en soustrayant au

fur et à mesure la chasse de chaque caractère. Cette méthode ne dissocie pas les signes à chasse nulle de leurs caractères de base car les signes qui les suivent n'ajoutent rien à la taille de la chaîne.

Si l'environnement textuel est plus perfectionné, il se peut que la chasse des caractères dépende du contexte en raison du crénage, des ligatures ou du choix d'une forme contextuelle. Pour ces systèmes, la chasse d'un caractère composé, comme le ï, peut être plus large que la chasse du caractère de base pris isolément. Pour prendre en compte ces cas, il est impératif de vérifier le résultat final tronqué par des soustractions successives.

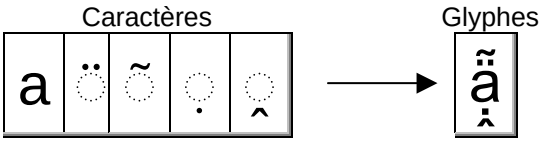
Une autre stratégie consiste à simplement rogner les caractères à l'affichage. Il se peut cependant que le résultat soit laid. Si le rognage se produit entre deux caractères, il se peut même qu'on ne se rende pas compte qu'il manque des caractères. Une image ou des points de suspension peuvent indiquer fort à propos cette omission.

5.14 Rendu des signes à chasse nulle

Cette section présuppose l'utilisation de polices proportionnelles pour lesquelles la chasse des différents caractères peut varier. On peut utiliser différentes techniques pour rendre les signes à chasse nulle à l'aide de police à chasse fixe, cependant d'ordinaire on ne parvient alors au mieux qu'à approcher un rendu acceptable, plus particulièrement dans le cas des caractères non latins.

Quand on affiche une suite constituée de plusieurs signes à chasse nulle, les signes doivent, par défaut, s'empiler l'un après l'autre à partir du caractère de base. En d'autres termes, si deux signes à chasse nulle surmontent un caractère de base, le premier signe doit apparaître juste au-dessus du caractère de base alors que le second signe domine le premier. Si les deux signes à chasse nulle se situent au pied d'un caractère de base, le premier signe doit apparaître juste en dessous du caractère de base et le second en dessous du premier (voir la *Section 2.6*, *Caractères combinatoires*). On appelle règle centrifuge ce comportement implicite de signes à chasse nulle (voir la *Figure 5-7*). Ces signes peuvent parfois interagir.

Figure 5-7. Règle centrifuge



Il est possible d'altérer le comportement par défaut en se basant sur des réglages typographiques particuliers ou sur les conventions typographiques régissant le rendu de plusieurs diacritiques d'une écriture particulière. Ainsi, en vietnamien moderne, les accents aigu et grave (ils indiquent le ton) peuvent être rendus légèrement de côté par rapport à un accent circonflexe plutôt que juste au-dessus. Si l'on sait (que cela soit implicitement en connaissant la langue du texte ou explicitement grâce à des étiquetes du texte enrichi), que le texte à afficher emploie une convention typographique différente, on peut alors rendre les signes à chasse nulle d'une manière qui contrevient à la règle centrifuge implicite.

Rendu de repli. Plusieurs méthodes permettent de gérer les suites de caractères composés inconnues qui n'appartient pas à un ensemble fixe affichable (voir la *Figure 5-8*). Une des méthodes (« révéler les diacritiques ») indique l'impossibilité de dessiner la suite en affichant d'abord le caractère de base pour ensuite rendre isolément le signe à chasse nulle – celui-ci s'affiche alors par rapport à un cercle en pointillés. (C'est la convention utilisée au *Chapitre 15*, *Tableaux de codes*.)

Figure 5-8. Rendu de repli

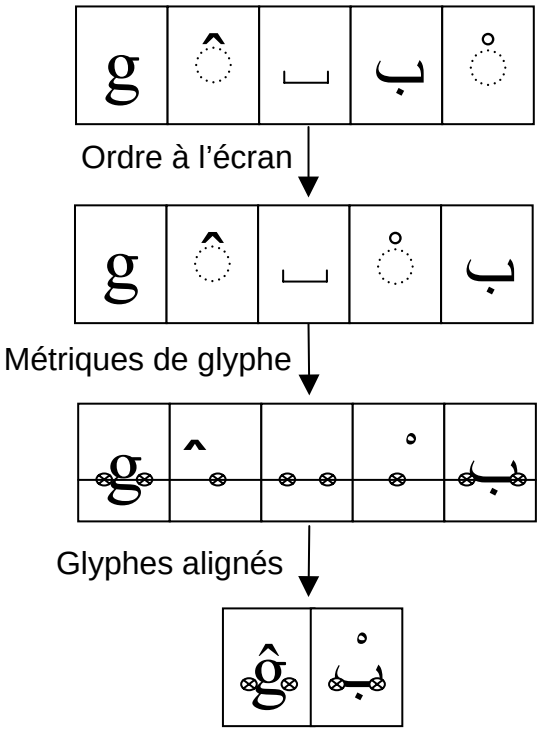
idéal | révéler les diacritiques | chevauchement élémentaire

Une autre méthode (« chevauchement élémentaire ») place tous les diacritiques à une position fixe, habituellement suffisamment éloignée des caractères de base. La position par défaut du circonflexe peut, par exemple, se situer au-dessus de l'ascendante, ce qui le place au-dessus des majuscules. Même si le résultat n'est guère esthétique pour des lettres comme *g-circonflexe,* on parviendra en général à reconnaître le caractère n'est muni que d'un seul diacritique.

Dans un cas dégénéré, le signe à chasse nulle se présente comme premier caractère d'un texte ou se trouve séparé de son caractère de base par un *séparateur de lignes*, un *séparateur de paragraphes* ou un autre caractère de formatage. Ce résultat constitue une *suite défectueuse de caractères combinatoires* (voir la *Section 3.5*, *Combinaison*). Il faut afficher les suites défectueuses de caractères combinatoires en considérant que leur caractère de base est une espace.

Positionnement bidirectionnel. Lors du réagencement des caractères bidirectionnels, les diacritiques accompagnent leur caractère de base. Ceci signifie qu'ils continuent à s'adjoindre visuellement aux mêmes caractères de base après l'utilisation de l'algorithme (cf. la *Figure 5-9*). Il existe quelques manières de s'assurer de ce positionnement.

Figure 5-9. Positionnement bidirectionnel



La méthode la plus simple s'apparente à la méthode de repli nommée « chevauchement élémentaire ». Dans l'algorithme bidirectionnel, les signes combinatoires héritent leur niveau directionnel de leur caractère de base. Dans ce cas-ci, les signes à chasse nulle arabes, hébreux ou syriaques se présenteraient donc à l'écran à gauche de leur caractère de base. On conçoit alors la police de sorte que les signes à chasse nulle droite-à-gauche saillent à droite plutôt qu'à gauche. Dans la *Figure 5-9*, les croix sur la ligne « métrique de glyphes » indiquent le début et la fin du tracé de chaque glyphe de ce type. Après l'alignement des points de départ et de fin, chaque signe à chasse nulle est attaché à sa lettre de base. Un rendu plus perfectionné peut ensuite recourir aux méthodes de positionnement brièvement exposées ci-dessous.

Certains logiciels de rendu exigent que les glyphes des signes à chasse nulle se placent toujours à la droite des glyphes correspondants à leur caractère de base alors que les diacritiques d'un texte bidirectionnel peuvent se situer à droite ou à gauche de leur caractère de base selon la directionalité de celui-ci (cf. la ligne « ordre à l'écran » de la *Figure 5-9*). Dans ce cas, il est possible d'effectuer une passe supplémentaire après la production de l'ordre visuel afin de placer les signes à chasse nulle de niveau impair (c'est-à-dire droite-à-gauche) à droite de leur caractère de base. (Cf. la *Section 3.12*, *Comportement bidirectionnel*).

Justification. En règle général, la justification en pavé d'un texte espace davantage les mots d'une ligne (on dit aussi « blanchit davantage la ligne») en ajoutant des espaces ou augmente l'interlettrage (voir la Figure 5-10). Il faut modifier ce processus quand un texte contient des signes à chasse nulle sans quoi ces signes se trouveraient isolés de leur caractère de base.

Figure 5-10. Interlettrage

66 points / 6 positions
= 11 points par position
66 points / 5 positions
= 13,2 points par position

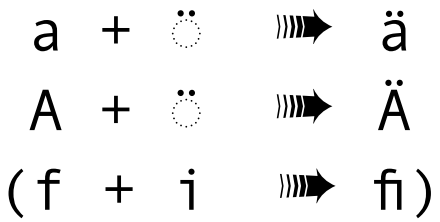
Les signes à chasse nulle suivent toujours leur caractère de base ; pour effectuer correctement une justification on n'ajoute donc de l'espace entre deux caractères qu'avant un caractère de base.

Méthodes de positionnement

Il existe différentes méthodes qui permettent de placer les signes à chasse nulle au bon endroit par rapport au caractère de base et aux signes à chasse nulle précédents.

Positionnement à l'aide de ligatures. On peut afficher de manière efficace un ensemble fixe de suites de caractères composés grâce à une substitution relativement simple (cf. *Figure 5-11*). Chaque fois que des glyphes représentent la suite <caractère de base, signe à chasse nulle>, on remplace la forme combinée par un nouveau glyphe correspondant. ٭tant donné que le diacritique a une chasse nulle, la suite de caractères composés aura automatiquement la même chasse que le caractère de base. (Des systèmes de rendu plus perfectionnés peuvent prendre en compte les cas spéciaux où la suite de caractères composés à une chasse légèrement différente du caractère de base.)

Figure 5-11. Positionnement à l'aide de ligatures

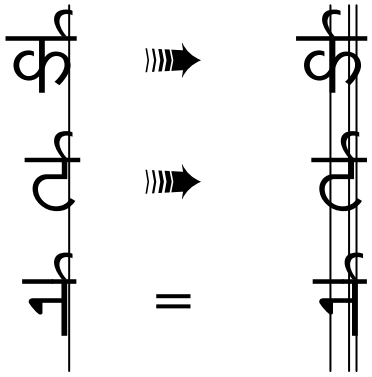


Le positionnement à l'aide de ligatures est sans doute la méthode la plus simple de prendre en charge les signes à chasse nulle. Chaque fois qu'il existe un petit ensemble fixe, comme celui qui correspond aux caractères précomposés de l'ISO 8859-1 (Latin-1), cette méthode est facile à implanter. Les séquences de caractères composés ont pratiquement toujours la même chasse que le caractère de base, ceci simplifie considérablement le rendu, la mesure et l'édition de ces caractères par rapport au cas plus général des ligatures.

Si une suite de caractères composés ne forme pas une ligature, on utilisera alors une des deux méthodes décrites ci-dessous. Si ces méthodes ne sont pas présentes, il faut alors se résoudre à utiliser une des méthodes de repli.

Positionnement à l'aide de formes contextuelles. Une méthode plus générale qui permet de placer correctement les signes à chasse nulle consiste à utiliser des glyphes contextuels (v. la *Figure 5-12* où l'on a recours à 3 glyphes différents pour les 3 positions du). Ces différents glyphes correspondent alors aux différentes positions des diacritiques. Les glyphes de base appartiennent habituellement à un petit nombre de classes, selon leur forme générale et leur taille. On choisit le glyphe du signe à chasse nulle en fonction du glyphe de base.

Figure 5-12. Positionnement à l'aide de formes contextuelles

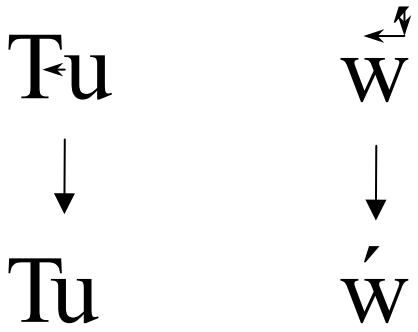


En règle générale, on a le choix entre différentes hauteurs de glyphe afin de pouvoir empiler les glyphes. Il est habituellement permis d'empiler de la sorte quelques diacritiques. Quand la limite est dépassée, on emploie une méthode de repli. La méthode à formes contextuelles peut être conjuguée à la méthode des ligatures afin de pouvoir, dans certains cas, utiliser des ligatures accentuées pour produire de subtiles variations dans la position et la forme des diacritiques.

Positionnement avec crénage optimisé. Une troisième technique de positionnement des diacritiques consiste à étendre le processus habituel du crénage afin qu'il soit à la fois horizontal et vertical (voir la *Figure 5-13*). D'ordinaire, une routine de crénage associe une valeur d'approche à une paire de glyphes. Ainsi, dans le mot « Tu », le « u » se niche-t-il

légèrement sous le « T ». Un système plus général consiste à associer des valeurs d'approche *horizontale* et *verticale* afin de modifier à volonté la position des glyphes.

Figure 5-13. Positionnement avec crénage optimisé



Pour que le cas général soit efficace, le processus de crénage doit également être capable de prendre en compte plus que de simples paires de glyphes puisque plusieurs diacritiques peuvent se présenter après la lettre de base.

Le positionnement avec crénage optimisé peut être conjugué à la méthode des ligatures afin de pouvoir, dans certains cas, utiliser des ligatures accentuées pour produire de subtiles variations dans la position et la forme des diacritiques.

5.15 Repérage des frontières d'élément textuel

Il arrive souvent qu'il faille diviser un texte Unicode en éléments textuels à l'aide d'un programme. Parmi les types d'élément textuel fréquents, on trouve ce que les utilisateurs considèrent comme des caractères, des mots, des lignes ou des phrases. La définition exacte de ces éléments textuels peut varier en fonction du profil local (« la locale ») même pour ce qui est de la définition d'un caractère. Il n'est pas toujours possible de respecter la perception que les utilisateurs ont de ces éléments car le texte même peut ne pas contenir suffisamment d'information pour déterminer les frontières de manière univoque. Ainsi, le point (U+002E . POINT) est-il polysémique, il indique parfois la fin d'une phrase, participe à la formation des abréviations ou sert de séparateur numéral. Toutefois, dans la plupart des cas, les frontières textuelles déterminées algorithmiquement se rapprochent fortement de celles perçues par l'utilisateur ou, du moins, elles ne le surprennent pas.

Plutôt que de se concentrer sur le repérage algorithmique des éléments textuels eux-même, il est plus simple de détecter les *frontières* entre ces éléments textuels. La détection de ces frontières est souvent essentielle pour le bon fonctionnement des logiciels d'utilité générale, il est donc important de pouvoir les déterminer aussi vite que possible.

Le mécanisme de repérage de frontière décrit ci-dessous permet de déterminer d'une manière simple et efficace les frontières de mots. Elle se base sur la représentation uniforme des caractères du standard Unicode tout en prenant en compte un grand nombre de caractères et de fonctionnalités spéciales comme les signes combinatoires et les seizets d'indirection[6]. Ce processus de repérage de frontières s'implante de façon naturelle et complète en se basant sur les données uniquement, il peut donc être personnalisé en fonction du profil local et plus

[6] Le traitement supplémentaire nécessaire pour prendre en compte les seizets d'indirection ne s'applique pas si on choisit une forme de stockage autre qu'UTF-16.

particulièrement la langue ou les besoins de l'utilisateur sans qu'on doive réécrire une ligne de code. Les frontières de mot, de ligne et de phrase doivent, plus particulièrement, être personnalisées en fonction de la langue et des préférences de l'utilisateur. En coréen, par exemple, on peut fractionner les lignes aux espaces (comme dans un texte latin) ou aux frontières idéographiques (comme en chinois).

Pour certaines langues, cette méthode aisée ne suffit pas. Ainsi en thaï, la coupure de lignes nécessite-t-elle un dictionnaire, par quoi elle est analogue à la coupure de mots en anglais. Une mise en œuvre devrait donc permettre que l'on redéfinisse ou sous-classe le processus rapide standard décrit ci-dessous dans la sous-section « Définition de frontière ».

La taille du jeu de caractères Unicode et sa puissance de représentation imposent des conditions à la fois sur la définition des frontières d'élément textuel et sur la mise en œuvre sous-jacente. La définition doit permettre de désigner d'importants ensembles de caractères qui partagent les mêmes caractéristiques (par exemple, les majuscules), alors que la mise en œuvre doit permettre d'accéder ou de comparer rapidement ces ensembles.

Le mécanisme proposé doit également prendre en compte les caractéristiques du standard Unicode comme les signes combinatoires ou à chasse nulle, les jamos jointifs ou les seizets d'indirection[6].

Le texte ci-dessous se penche sur deux aspects reliés aux frontières d'élément textuel : la définition et la mise en œuvre sous-jacente.

Définition de frontière

La définition d'une frontière établit différentes classes puis énumère les règles des frontières en fonction de ces classes. Il faut se rappeler qu'un caractère peut être représenté par une suite de deux seizets d'indirection en UTF-16. On exprime les classes de caractères sous la forme d'une liste, dont chaque élément peut représenter :

- un caractère littéral ;
- un intervalle de caractères ;
- la propriété d'un caractère Unicode, telle que définie dans *la Base de données des caractères Unicode* ;
- l'exclusion d'éléments de la liste.

Pour une description formelle de la notation utilisée dans cette section, veuillez consulter la *Section 0.2*, *Conventions de notation*. La présente section fait usage de quelques conventions supplémentaires :

÷ indique une position sécable

× indique une position insécable

_ un tiret bas indique une espace dans les exemples.

On associe toujours une frontière implicite au début et à la fin d'un texte. Comme c'est le cas avec les expressions régulières typiques, on utilise la plus longue production possible. Ainsi, dans l'exemple suivant, la frontière est placée après le maximum de Y permis :

XY* ÷ ¬X

(À des fins de référence, les règles de cet ouvrage sont numérotées.) La définition de frontière obéit à quelques contraintes supplémentaires. Ces contraintes simplifient considérablement et

rendent nettement plus efficace la mise en œuvre sans apparemment induire de restrictions pour les langues naturelles.

1. Contexte limité. Soit deux frontières aux positions X et Y, la position des autres frontières entre X et Y ne dépend pas des caractères situés à l'extérieur de X et Y (pour autant que X et Y demeurent des positions de frontière).

Admettons, par exemple, des frontières[7] à « ab÷cde », modifier « a » en « A » n'introduit pas une nouvelle frontière entre d et e de telle sorte qu'on ait alors « Ab÷cd÷e ».

2. Frontières simples. Chaque règle ne décrit qu'une position de frontière. Étant donné la règle (1), cette restriction constitue plus une limitation dans la manière de spécifier les règles, puisqu'une règle avec deux frontières peut habituellement s'exprimer sous la forme de deux règles.

Exemple : la règle « ab÷cd÷ef » peut être subdivisée en « ab÷cd » et « cd÷ef ».

3. Sans conflit. Deux règles ne peuvent partager des portions initiales qui correspondent au même texte mais aux positions de frontière différentes.

Exemple : « x÷abc » et « a÷bc » ne peuvent faire partie de la même définition de frontière.

4. Sans recouvrement d'ensembles. Pour des raisons d'efficacité, deux ensembles de caractères dans une définition ne peuvent se recouvrir. La dernière définition d'ensemble de caractères a précédence sur la définition antérieure et élimine les caractères communs de l'ensemble antérieur.

Dans l'exemple ci-dessous, la définition du second ensemble élimine « AEIOUaeiou » du premier :

Let = [Lu][Ll][Lt][Lm][Lo]

VoyellesNonAccentuées = AEIOUaeiou

5. Pas plus de 256 ensembles. L'objection de cette restriction est de produire des mises en œuvre compactes.

6. Ignorer les dégénérés. Les mises en œuvre ne doivent pas tenter d'améliorer l'algorithme uniquement pour des cas dégénérés qui, en pratique, ne se présentent jamais : un A suivi d'un signe combinatoire hindi.

On peut approcher ces règles à l'aide de tableaux de paires qui ne considèrent que le caractère qui précède et celui qui suit un point de coupure potentiel. Bien que cette méthode ne produise pas, en règle générale, les résultats attendus, elle suffit souvent si on limite les langues considérées. Néanmoins, les frontières de graphème sont habituellement conçues pour pouvoir s'exprimer complètement à l'aide de tableaux de paires.

Définitions modèles

Comme nous le verrons dans la section ci-dessous, chaque type de frontière est associé à une série de considérations différentes. Dans cette section, nous simplifierons quelque peu les règles de frontière et nous ne prendrons pas en considération tous les cas limites. Les caractères de formatage et de commande, notamment, ne sont pas des frontières de coupure. Prendre en considération de tels cas compliquerait indûment chacun des exemples présentés

[7] Les frontières sont situées au début du texte, au ÷ (X) et à la fin du texte (Y).

(ceci est toutefois laissé à titre d'exercice aux lecteurs). Ces règles, de surcroît, ont été conçues pour qu'elles soient localisables. Les exemples fournis ci-dessous ne sont pas valables pour tous les profils locaux.

On expliquera le contenu des ensembles de caractères utilisés dans ces exemples, plutôt que d'en énumérer les éléments.

Si une mise en œuvre utilise une table d'état, sa performance ne dépend ni de la complexité ni du nombre de règles. La seule variable qui influence la performance est le nombre de caractères à considérer situés *après* la frontière d'une règle candidate.

Frontières de graphème

Un *graphème* est la plus petite unité de sens d'un système d'écriture, tout comme un phonème est la plus unité de sens dans la chaîne parlée. Parfois, un graphème se représente à l'aide de plus d'un caractère Unicode, comme c'est le cas dans les langues de l'Inde. En ce qui concerne l'utilisateur, la représentation sous-jacente du texte est sans importance, mais il est crucial que les interfaces d'édition mettent en œuvre de façon uniforme ce que l'utilisateur considère être un graphème. Les graphèmes doivent se comporter comme des unités lors de la sélection, du déplacement à l'aide des flèches ou de l'utilisation de la touche « supprimer ». Ainsi, si un caractère accentué est représenté par une suite de caractères combinatoires, alors appuyer sur la « flèche à droite » devra déplacer le curseur du début du caractère de base à la fin du dernier caractère combinatoire antérieur. Cette situation s'apparente à celle d'un système qui utilise des jamos jointifs pour représenter des syllabes hangûls et qui, lors de l'édition, traite ces syllabes comme s'il s'agissait de graphèmes isolés. Quand l'utilisateur final désire connaître le nombre de caractères (ce qui en pratique est assez rare), ce nombre doit correspondre à la perception de ce que l'utilisateur considère être un graphème.

Le traitement correct des signes combinatoires, des jamos jointifs hangûls et des groupes de caractères tibétains et de l'Inde nécessite la définition des frontières de caractère. Voir le *Tableau 5-3*.

Tableau 5-3. Frontières de graphème

Classes de caractères	
CR	Retour de chariot
LF	Passage à la ligne
Format	Tous les autres caractères de formatage ou de commande
Virama	Les viramas indiens
Jointif	Tous les caractères combinatoires, ainsi que les subjointes tibétaines
I	Jamo hangûl initial
V	Jamo voyelle hangûl
F	Jamo hangûl final
Lo	Autres lettres
Autres	Tous les autres caractères

Règles

Toujours couper devant un LF, à moins qu'il ne soit précédé d'un CR. Toujours couper avant et après tous les autres caractères de formatage ou de commande.

¬CR	**÷**	**LF**	**(1)**
CR	**÷**	**¬LF**	**(2)**
	÷	**(CR | Format)**	**(3)**
(Format | LF)	**÷**		**(4)**

Couper devant les viramas et les autres caractères jointifs uniquement si ceux-ci sont précédés de caractères de formatage ou de commande.

(Format | CR | LF)	**÷**	**(Virama | Jointif)**	**(5)**

Couper devant et après un jamo hangûl à moins qu'il ne forme une suite permise.

¬I	**÷**	**I**	**(6)**
¬(I | V)	**÷**	**V**	**(7)**
¬(I | V | F)	**÷**	**F**	**(8)**
(I | V | F)	**÷**	**¬(I | V | F)**	**(9)**

Ne pas couper entre les viramas et les autres lettres. Cette règle permet de prendre en compte les graphèmes indiens, où le virama lie les groupes de caractères.

Virama	**×**	**Lo**	**(10)**

Couper avant tout autre caractère.

	÷	**Autres**	**(11)**

Frontières de mot

Les frontières de mot s'utilisent dans différents contextes. Les plus familiers sont la sélection par un double clic de la souris, la fonction « passer au mot suivant » et la détection de mots complets lors d'une recherche et remplacement.

En ce qui concerne l'option de recherche et remplacement appelée « trouver un mot complet », les règles sont relativement claires. Les frontières se situent entre les lettres et les non-lettres. Il faut exclure les espaces de queue de la définition d'un mot car, confronté à "abc_", la recherche de "abc" échouerait alors.

Dans le cas de la sélection de mots, les règles sont quelque peu moins précises. Certains programmes admettent les espaces de queue, alors que d'autres admettent la ponctuation qui jouxte le mot. Quand les mots ne comprennent pas d'espace de queue, les programmes considèrent parfois les espaces comme autant de mots, alors que d'autres considèrent une suite contiguë de blancs comme un seul mot (cette dernière méthode s'accorde mieux avec l'utilisation de la recherche et remplacement).

- On peut également utiliser les frontières de mot dans ce qu'on est convenu d'appeler le « copier-coller intelligent ». Grâce à celui-ci, si un utilisateur coupe un morceau de texte aux frontières de mots, les espaces multiples soudainement adjacents se voient remplacer

par un seul. Ainsi, extraire "vieux" de "Apportez_ce_vieux_whisky" produit habituellement "Apportez_ce_ _whisky". Un copier-coller intelligent réduit ce texte à "Apportez_ce_whisky".

Cette section décrit à grands traits le cas où les frontières apparaissent entre les lettres et les non-lettres et où il n'existe pas de frontière entre les non-lettres. Elle prend également en considération les mots japonais (pour la sélection de mots). En japonais, il faut le rappeler, les mots ne sont pas séparés par des espaces. On utilise plutôt une règle heuristique pour établir la frontière de mots. Celle-ci considère les mots comme des chaînes de *kanji* (idéogrammes) ou de caractères katakana (qui peuvent être suivis d'une chaîne de caractères hiragana). Voir le *Tableau 5-4*.

Tableau 5-4. Frontières de mot

Classes de caractères

CR	Retour de chariot
LF	Passage à la ligne
Sép	Séparateur de lignes ou de paragraphe
TAB	Caractère de tabulation horizontale
Let	Lettre
Com	Signe combinatoire
Hira	Hiragana
Kata	Katakana
Han	Idéogramme han (kanji, han-tseu)

Règles

Toujours couper devant un LF, à moins qu'il ne soit précédé d'un CR. Toujours couper avant et après tous les autres caractères de formatage ou de commande, y compris TAB.

¬CR	**÷**	**LF**	**(1)**
CR	**÷**	**¬LF**	**(2)**
	÷	**(CR | Sép | TAB)**	**(3)**
(Sép | TAB | LF)	**÷**		**(4)**

Couper entre les lettres et les non-lettres. Considérer les signes à chasse nulle qui suivent directement une lettre comme en faisant partie.

¬(Let | Com)	**÷**	**Let**	**(5)**
Let Com*	**÷**	**¬Let**	**(6)**

Pour la sélection de mot, traiter le japonais à part. Traiter les groupes de kanjis ou de katakanas (suivis ou non de hiraganas) comme des mots. Couper s'ils sont précédés d'autres caractères (p. ex. des signes de ponctuation). Inclure les signes à chasse nulle.

¬(Hira | Kata | Han | Com)	**÷**	**Hira | Kata | Han**	**(7)**
Hira Com*	**÷**	**¬Hira**	**(8)**
Kata Com*	**÷**	**¬(Hira | Kata)**	**(9)**
Han Com*	**÷**	**¬(Hira | Han)**	**(10)**

Il est également permis en règle générale de couper entre les lettres d'écritures différentes. En pratique, sauf pour des écritures qui n'utilisent pas d'espace, cette possibilité constitue un cas dégénéré.

Frontières de ligne

Les frontières de ligne identifient les endroits où le renouement de lignes peut se produire sans nécessiter de coupure de mots. (Les renouement de lignes perfectionnés utilisent souvent la coupure de mots mais uniquement lorsque le repli de ligne naturel produit un résultat inacceptable). Bien que cette notion soit fort similaire à celle de la frontière de mots, en règle générale elle n'y est pas identique.

Pour les fins du renouement de lignes, on considère en général qu'une suite de caractères composés possède les mêmes propriétés que son caractère de base. Il est impératif de ne pas séparer ces signes à chasse nulle de leur caractère de base.

Un signe à chasse nulle peut s'afficher sous forme isolée — en d'autres mots, adjoint à une espace sécable ou insécable. Dans ce cas, on traite la suite complète de caractères composés de manière atomique. Si la suite de caractères composés commence par une *espace insécable,* on ne permet pas alors, en général, de couper la ligne juste avant ou après cette suite. Si la suite de caractères composés commence par toute autre espace, il est alors permis de replier la ligne juste avant ou après cette suite.

Le cas suivant est dégénéré : un signe à chasse nulle commence un texte ou suit immédiatement un séparateur de lignes ou de paragraphes. Dans ce cas, le traitement le plus cohérent consiste à considérer que ce signe à chasse nulle s'adjoint à une espace implicite. Voir le *Tableau 5-5*.

Tableau 5-5. Frontières de ligne

Classes de caractères	
CR	Retour de chariot
LF	Passage à la ligne
ESPSC	Espace à chasse nulle
ESPINSC	Espace insécable à chasse nulle
Esp	Espaces
Coupure	Coupure obligatoire (par exemple, un séparateur de paragraphes)
Com	Tous les signes combinatoires (y compris les subjointes tibétaines), en plus des jamos jointifs médians et finaux hangûls.
Idéographique	Caractères idéographiques
Alphabétique	Caractères alphabétiques et la plupart des symboles
Exclam	Caractères terminaux comme le point d'exclamation
Syntaxe	Barre oblique (« / »)
Ouvrante	Ponctuation ouvrante, comme le « (»
Fermante	Ponctuation fermante, comme le «) »
Guille	Guillemets ambigus comme les « , les " ou les ‘
NonAmorce	Petits caractères hiragana ou katakana

TiretMoins	U+002D
Insép	Points de conduite ou de suspension
Nombre	Chiffres
PréfixeNum	Caractères comme le « + » ou « $ »[8]
PostfixeNum	Caractères comme le « % »
InfixeNum	Signes que l'on rencontre au sein des nombres européens
Base	Caractères de base
NonBase	Caractères qui ne sont pas de base (caractères de commande et de formatage)
Autres	Tous les autres caractères Unicode

Remarque : pour une définition précise de ces classes, veuillez consulter le rapport technique Unicode n° 14 *Line Breaking Properties* présent sur le disque qui accompagne cet ouvrage ou sur le site Unicode pour une version tenue à jour. Les classes ci-dessus ont des noms quelque peu différentes pour des raisons de cohérences.

Règles

Coupures et non-coupures explicites :

- Toujours couper après un passage à la ligne explicite (mais jamais entre CR et LF). On peut choisir de couper après chaque ESPSC, mais pas après un passage à la ligne explicite.
- Ne pas couper avant les espaces ou les passages à ligne explicites.
- Ne pas couper avant ou après un ESPINSC.

CR	×	**LF**	**(1)**
(CR | LF | Coupure | ESPSC)	÷		**(2)**
	×	**(Esp | CR | LF | Coupure | ESPSC | ESPINSC)**	**(3)**
ESPINSC	×		**(4)**

Signes combinatoires :

- Ne pas séparer les graphèmes de leur signes combinatoires qui suivent. Les viramas et les jamos sont attribués aux classes appropriées afin que leur coupure soit correcte.

Dans toutes les règles suivantes :

- Si une espace constitue le caractère de base d'un signe combinatoire, le type de cette espace devient AL (alphabétique).
- Chaque fois que l'on peut couper entre un Com et un caractère qui le suit, le Com se comporte comme s'il était du même type que son caractère de base. S'il n'existe pas de caractère de base, le Com se comporte comme un AL.

[8] Ces classes sont prévues pour les conventions typographiques anglo-saxonnes, rappelons qu'elles doivent être adaptées au profil local de l'utilisateur (« localisées »).

	×	**Com**	**(5)**
Esp Com	→	**Alphabétique Com**	**(6)**
Base Com	→	**Base Base**	**(7)**
NonBase Com	→	**NonBase Alphabétique**	**(8)**

Traitement des signes ouvrants et fermants :

Ils se comportent d'une manière particulière en ce qui a trait aux espaces.

- Ne pas couper avant les points d'exclamation, les caractères syntaxiques ou les signes de ponctuation fermante, même après des espaces.
- Ne pas couper avant des signes de ponctuation ouvrante, même après des espaces.
- Ne couper ni entre des guillemets et des signes de ponctuation ouvrante, ni entre des marques de ponctuation fermante et des idéogrammes, même séparés d'espaces.

	×	**(Exclam | Syntaxe | Fermante)**	**(9)**
Ouvrante	×		**(10)**
Guille	×	**Ouvrante**	**(11)**
Fermante	×	**Idéographique**	**(12)**

Une fois les marques ouvrantes et fermantes traitées, on peut en terminer avec les espaces.

- Couper après les espaces.

Esp	÷	**(13)**

Cas particuliers :

- Ne pas couper avant ou après des guillemets ambigus.
- Ne pas couper avant des NonAmorces ou un TraitMoins.
- Ne couper ni entre deux séries de points de suspension ni entre des lettres ou des nombres et des points de suspension, comme « 9… », « a… » ou « H… ».
- Ne pas séparer les caractères alphabétiques des chiffres, les chiffres des caractères alphabétiques ou les idéogrammes des suffixes numériques**.**

	×	**(Guille | NonAmorce | TiretMoins)**	**(14)**
Guille	×		**(15)**
(Insép | Nombre | Idéographique | Alphabétique)	×	**Insép**	**(16)**
Alphabétique	×	**Nombre**	**(17)**
Nombre	×	**Alphabétique**	**(18)**
Idéographique	×	**PostfixeNum**	**(19)**

Les nombres peuvent prendre différentes formes qu'il faut conserver sur une même ligne – par exemple, 12,54 $, 1 000 £, (12)¢, $(12.35) et ainsi de suite.

Les règles suivantes représentent approximativement ce comportement (on a déjà traité certains cas comme « 9 » ou « [9 »).

PréfixeNum	×	**(Nombre | Ouvrante | TiretMoins)**	**(20)**
(TiretMoins | Syntaxe | Nombre | InfixeNum)	×	**Nombre**	**(21)**
Nombre	×	**PostfixeNum**	**(22)**
Fermante	×	**PostfixeNum**	**(23)**

Relier les caractères alphabétiques, couper tout le reste.

Alphabétique	×	**Alphabétique**	**(24)**
	÷	**Autres**	**(25)**
Autres	÷		**(26)**

Frontières de phrase

On utilise souvent les frontières de phrase pour mettre en œuvre le triple clic ou toute autre méthode de sélection ou d'itération qui opère sur des morceaux de texte plus importants que les mots.

Un texte brut ne fournit pas suffisamment d'information pour permettre de détecter les bonnes frontières de phrase. Les points, par exemple, peuvent indiquer la fin d'une phrase, une abréviation ou servir de séparateur numéral. Sans analyse sémantique du texte, il est impossible de déterminer avec certitude l'utilisation du point (et même dans ce cas, il demeure parfois des ambiguïtés). Voir le *Tableau 5-6*.

Tableau 5-6. Frontières de phrase

Classes de caractères	
CR	Retour de chariot
LF	Passage à la ligne
Sép	Séparateur de lignes ou de paragraphe
Term	!?
Point	Lettre
Maj	Majuscules, casse de titre et lettres sans casse
Min	Minuscules
Ouvrante	Ponctuation ouvrante
Fermante	Ponctuation fermante : point, virgule, etc.

Règles

Toujours couper avant un LF à moins qu'il ne soit précédé d'un CR. Toujours couper avant et après les caractères de commande ou de formatage séparateurs.

¬CR	**÷**	**LF**	**(1)**
CR	**÷**	**¬LF**	**(2)**
	÷	**(CR | Sép)**	**(3)**
(Sép | LF)	**÷**		**(4)**

Couper après les terminateurs de phrase, y adjoindre les signes sans chasse, de ponctuation fermante, les espaces terminales et, facultativement, un séparateur de paragraphes.

Term Fermante* Esp* {Sép}	**÷**		**(5)**

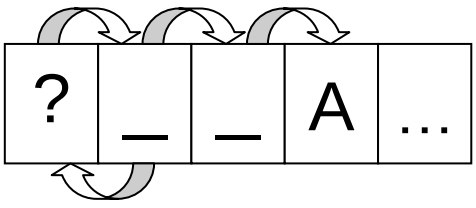
Traiter le point final à part puisqu'il peut s'agir d'un point numéral ou d'abrègement — et non de la fin d'une phrase. Ne pas couper si le point est suivi d'une minuscule plutôt que d'une majuscule.

Point Fermante* Esp+	**÷**	**Ouvrante* ¬Min**	**(6)**

Accès direct

L'accès direct d'une chaîne entraîne des complications supplémentaires (voir la *Figure 5-14*). Quand on itère au travers d'un chaîne, du début à sa fin, la méthode décrite ci-dessus fonctionne bien. Elle garantit un contexte limité et permet, à chaque frontière, de recommencer à neuf la recherche de la frontière suivante. On peut effectuer les recherches inverses en construisant une table à états pour la direction inverse correspondant aux mêmes règles. Supposons, toutefois, que l'utilisateur veuille itérer à partir d'un point quelconque du texte. Si le point de départ (aléatoire) ne fournit pas suffisamment de contexte pour permettre l'application des bonnes règles, il se peut alors qu'on ne repère pas une frontière valable. Ainsi, supposons qu'un utilisateur clique après la première espace dans "?_ _ A". Si on utilise la méthode itérative avant pour localiser la frontière de phrase, on ratera la frontière située avant le « A » puisque le « ? » n'a pas encore été aperçu.

Figure 5-14. Accès direct



Pour résoudre ce problème on établit un autre ensemble de règles dont l'objectif est de déterminer les points de départ « sûrs ». On rebrousse chemin vers le début de la chaîne en appliquant ces règles jusqu'à ce qu'on trouve un point de départ sûr. On peut alors itérer vers la fin de la chaîne à partir d'ici. Lors de cette itération avant, on ignore toutes les frontières situées entre le point sûr et le point de départ. La frontière recherchée est la première qui ne précède pas le point de départ.

Ce processus grèverait la performance s'il devait être effectué pour chaque recherche. Cependant, cette fonctionnalité peut être encapsulée dans un objet itérateur qui conserve l'information nécessaire pour déterminer si on se trouve actuellement à un point de frontière valable. Ce n'est que dans le cas où on se repositionne à un point arbitraire du texte qu'il faudra recourir à nouveau à ce processus d'appoint.

5.16 Identificateurs

Une des tâches fréquentes à laquelle est confrontée la personne chargée de la mise en œuvre d'Unicode est la réalisation de routines d'analyse syntactique et lexicale des identificateurs. Afin de permettre un traitement uniformisé des identificateurs par les analyseurs utilisant des caractères Unicode, nous fournissons ci-dessous quelques conseils relatifs à la définition syntaxique des identificateurs. Veuillez également vous reporter à la *Section 6.12*, *Identificateurs dans les langages informatiques*.

Soulignons que l'analyse des identificateurs et le repérage des frontières de mots sont des tâches apparentées. Il est aisé de reformuler la syntaxe de référence présentée ci-dessous en utilisant le formalisme dans le *Tableau 5-4.* Nous utiliserons dans cette section la syntaxe BNF plus traditionnelle afin de faciliter son incorporation dans les normes existantes.

L'objectif de la syntaxe formelle présentée ici est de décrire le fait qu'un identificateur se compose d'une chaîne de caractères qui commence par une lettre ou un idéogramme suivi d'un nombre quelconque de lettres, d'idéogrammes, de chiffres ou de traits soulignés. Chaque langage de programmation a sa propre syntaxe pour les identificateurs. Chaque langage à ses propres conventions concernant l'utilisation de certains caractères ASCII ($, @, #, _) au sein des identificateurs. Afin de compléter cette syntaxe et de mettre en œuvre Unicode, il suffit aux développeurs de fusionner les règles existantes de leur analyseur et la syntaxe de référence présentée ci-dessous.

Ces règles ne sont pas plus complexes que les règles actuelles des grands langages de programmation, si ce n'est qu'elles acceptent plus de caractères de types différents.

La syntaxe de référence des identificateurs Unicode introduit les fonctions suivantes absentes des analyseurs pré-Unicode :

1. Le traitement correct des signes combinatoires
2. La tolérance aux caractères de commande de composition et de formatage, à ignorer lors de l'analyse des identificateurs.

Signes combinatoires. Il faut prendre en compte les signes combinatoires dans la syntaxe des identificateurs. Une suite de caractères composés (constituée d'un caractère de base et d'un nombre arbitraire de signes combinatoires) est un identificateur valable. Cette exigence découle des règles de conformité du *Chapitre 3, Conformité*, relatives à l'interprétation des suites de caractères canoniquement équivalentes.

Les caractères combinatoires englobants (par exemple, U+20DD ◌⃝ DIACRITIQUE CERCLE ENGLOBANT à U+20E0 ◌⃠ DIACRITIQUE CERCLE ENGLOBANT ET BARRE OBLIQUE INVERSÉE) sont exclus de la définition syntaxique de la <suite_identificateur> car les caractères composites issus de leur composition avec des lettres (par exemple, 24B6 Ⓐ LETTRE MAJUSCULE LATINE A CERCLÉE) sont eux-mêmes exclus comme éléments syntaxiques d'un identificateur.

Caractères de commande de composition et de formatage. Les caractères Unicode utilisés pour commander le comportement jointif, l'ordre bidirectionnel et les formats de substitution n'ont pas d'effet sur la coupure. Contrairement aux espaces ou à d'autres délimiteurs, ils ne servent pas à indiquer la frontière de mots, de lignes ou de toute autre unité. C'est pourquoi on

les inclut explicitement dans la définition des identificateurs. Certaines mises en œuvre pourront choisir d'éliminer ces caractères « ignorables », cette méthode présente l'avantage de traiter plus souvent de manière identique deux identificateurs qui paraissent être identiques.

Addition de caractères particuliers. On peut considérer la syntaxe des identificateurs particulière à un langage comme une légère modification de la syntaxe de référence définie à partir des propriétés de caractère. Ainsi, par exemple, les identificateurs SQL acceptent-ils un tiret bas (ou souligné) au sein d'un identificateur (mais non au début de celui-ci) alors que les identificateurs C admettent ce tiret bas au début de l'identificateur et dans sa suite.

La notation utilisée dans cette section est exposée à la *Section 0.2*, *Conventions de notation*.

Règle syntaxique

<identificateur> : := <début_identificateur> (<début_identificateur> |
<suite_identificateur>)*

On définit les identificateurs à l'aide d'un ensemble de catégories de caractères spécifiées dans la *Base de données de caractères Unicode*. Voir le *Tableau 5-7*.

Les mises en œuvre qui nécessitent une définition stable des identificateurs pour des raisons de conformité doivent se référer à une version précise du standard Unicode (par exemple, la version 3.0.0), ou énumérer explicitement le sous-ensemble des caractères Unicode qui correspond à leur définition des classes syntaxiques.

Tableau 5-7. Classes syntaxique des identificateurs

Classe syntaxique	Catégories équivalentes	Domaine couvert
<début_identificateur>	Lu, Ll, Lt, Lm, Lo, Nl	Lettres majuscules, minuscules, de casse de titre, modificatives, autres lettres et lettres numérales.
<suite_identificateur>	Mn, Mc, Nd, Pc, Cf	Signes sans chasse, signes combinatoires avec chasse, chiffres décimaux, ponctuation de connexion, code de formatage.

5.17 Tri et repérage

Le tri et le repérage se recoupent car tous deux mettent en œuvre des degrés d'*équivalence* entre des termes à comparer. Dans le cas du repérage, l'équivalence définit la correspondance entre deux termes (elle définit, par exemple, les cas où la différence de casse est significative). Pour le tri, l'équivalence définit la proximité des termes d'une liste triée. Ces équivalences dépendent toujours de l'application et de la langue. Cependant une mise en œuvre qui prend en charge le standard Unicode, le tri et le repérage doit considérer les équivalences de caractère et les formes de normalisation Unicode. Voir les chapitres 3, *Conformité*, et 6, *Formes de normalisation*.

Cette section aborde également les problèmes liés à l'adaptation des algorithmes sublinéaires de repérage de texte, adin qu'ils permettent une recherche textuelle rapide sensible aux usages linguistiques tout en utilisant le même algorithme de mise en ordre que le tri. Pour plus d'informations, veuillez vous référer au rapport technique Unicode n° 10 *Unicode Collation Algorithm* présent sur le disque qui accompagne cet ouvrage ou, pour une version tenue à jour, sur le site Internet du consortium Unicode.

Tri conforme à l'usage culturel

Chaque culture trie différemment les chaînes de caractères. De surcroît, de nombreuses applications particulières nécessitent de modifier quelque peu l'ordre local. Un tri peut s'effectuer en fonction des mots ou des phrases, peut considérer ou non la casse ou encore les accents. Il peut également être phonétique ou effectué en fonction de l'apparence du caractère, comme c'est le cas pour le tri selon la clé et le nombre de traits utilisés pour les idéogrammes de l'Extrême-Orient. Le tri phonétique des caractères han nécessite la consultation d'un dictionnaire ou de programmes spéciaux qui associent une annotation phonétique à chaque caractère du texte.

Les langues ne varient pas uniquement par le type de tris qu'elles utilisent (et l'ordre dans lequel ceux-ci sont exécutés), mais également par leur définition de ce qui constitue un élément fondamental de tri. Ainsi, le suédois considère U+00C4 Ä LETTRE MAJUSCULE LATINE A TRÉMA comme étant une lettre à part, triée après *z* dans l'alphabet. L'allemand, par contre, trie cette lettre comme s'il s'agissait d'un ae ou d'une forme accentuée de la lettre de base et, en conséquence, la place à la suite du *a*. En espagnol, on trie traditionnellement le digramme *ll* comme s'il s'agissait d'une lettre entre *l* et *m*. Les exemples pour d'autres langues (et écritures) sont légion.

En conséquence, il est impossible de ranger les caractères d'un jeu de sorte qu'une simple comparaison binaire de chaînes produise l'ordre lexicographique désiré, il est donc également impossible de produire des tableaux de pondération de tri à un seul niveau. Ceci signifie que la manière précise dont sont codés les caractères n'a qu'une influence indirecte sur un tri conforme à l'usage culturel.

Pour résoudre les complexités inhérentes au tri conforme à l'usage culturel, on emploie typiquement un algorithme de comparaison multiniveau. On associe à chaque caractère d'une chaîne plusieurs catégories de *poids de tri*. Ces catégories peuvent correspondre à des poids associés aux lettres, aux diacritiques, ou à la chasse, etc.

Lors de la première passe, on accumule ces poids pour former une *clé de tri* associée à la chaîne. À la fin de cette passe, la clé de tri est formée d'une chaîne de poids alphabétiques, suivie d'une chaîne de poids de casse et ainsi de suite. Lors de la seconde passe, on compare ces sous-chaînes par ordre d'importance de telle sorte que les différences de casse ou de diacritiques peuvent être complètement ignorées ou n'être prises en compte que dans les cas où il faut différencier des chaînes de tri qui seraient autrement identiques.

La première passe ressemble fort à la décomposition des caractères Unicode en leur caractère de base et leurs accents. Le fait que le standard Unicode permette de coder de plusieurs façons une même lettre accentuée (précomposée ou composée) n'a en fin de compte pas d'importance. En fait, un texte complètement décomposé simplifie plutôt une première mise en œuvre du tri.

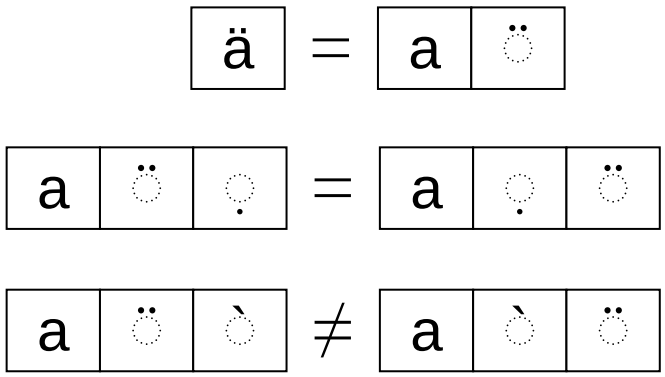
L'implantation d'un tri Unicode rapide à l'aide d'un tableau impose la prise en charge complète des fonctions suivantes caractéristiques d'un tri algorithmique sensible à la langue :

- Quatre niveaux de tri ;
- Sens français ou étranger ;
- Contraction et expansion des caractères ;
- Tri des caractères non pertinents ou sans correspondance ;
- Plus d'un niveau de caractères « ignorables ».

Équivalence de caractères Unicode

Les sections *3.6*, *Décomposition*, et 3.10, *Mise en ordre canonique*, ainsi que le *Chapitre 6*, *Formes de normalisation*, définissent des suites équivalentes et fournissent un algorithme précis qui permet de déterminer l'équivalence de deux suites. Les suites Unicode équivalentes doivent être triées d'une manière strictement identique, quelle que soit la manière dont elles sont stockées. La *Figure 5-15* illustre deux types d'équivalence de caractères.

Figure 5-15. Équivalence de caractères



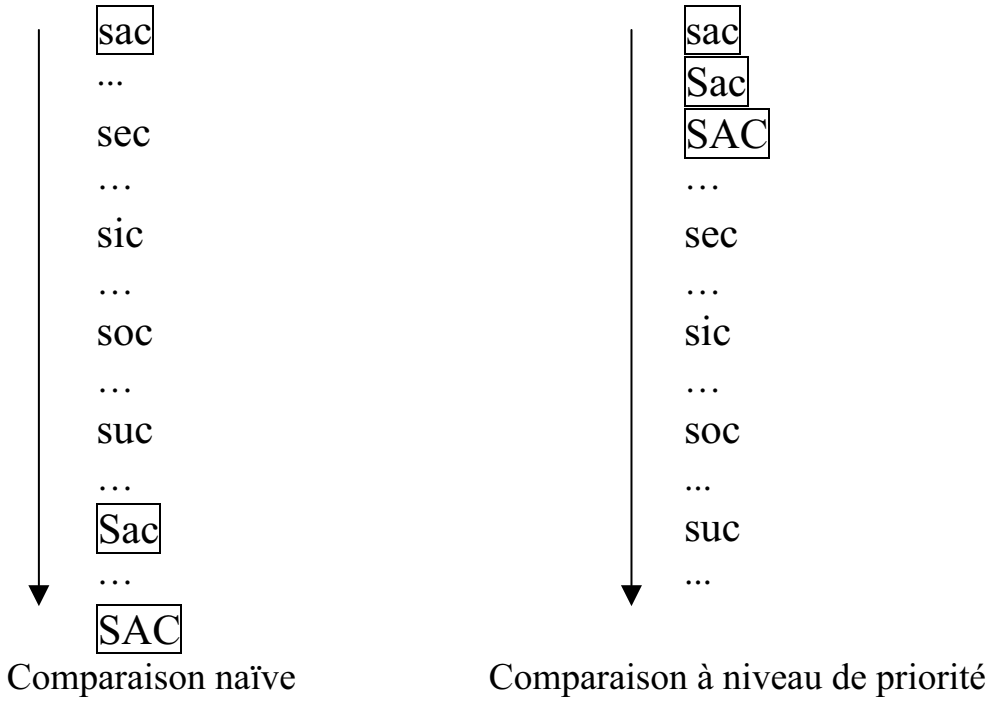
Les caractères de compatibilité — plus particulièrement quand ils ont le même dessin — devraient être trié d'une manière strictement identique. C'est le cas, par exemple, pour U+00C5 Å LETTRE MAJUSCULE LATINE A ROND EN CHEF et U+212B Å SYMBOLE ANGSTRÖM.

Caractères similaires

Les langues ne s'accordent pas sur ce qui constitue des caractères similaires. Cependant les utilisateurs désirent habituellement que des caractères similaires (comme les minuscules et majuscules) soient triés à proximité l'un de l'autre sans se confondre. Si les majuscules et les minuscules se classent de manière identique, les mots qui ne diffèrent que par leur casse apparaîtront dans un ordre apparemment aléatoire (voir la *Figure 5-16*). Il en va de même pour les caractères accentués.

D'ordinaire, il est cependant possible de classer ces caractères similaires. Dans les dictionnaires anglais, par exemple, les minuscules précèdent les majuscules (contrairement à une comparaisons ASCII naïve). Cependant ce classement n'intervient que lorsque les chaînes sont égales sous tout autre rapport, sans cela *Aachen* (Aix-la-Chapelle) serait trié après *azure*. Les caractères accentués se trient souvent près de leur caractère de base, mais différents accents sur un même caractère de base se classe toujours dans le même ordre.

Figure 5.16. Comparaison naïve



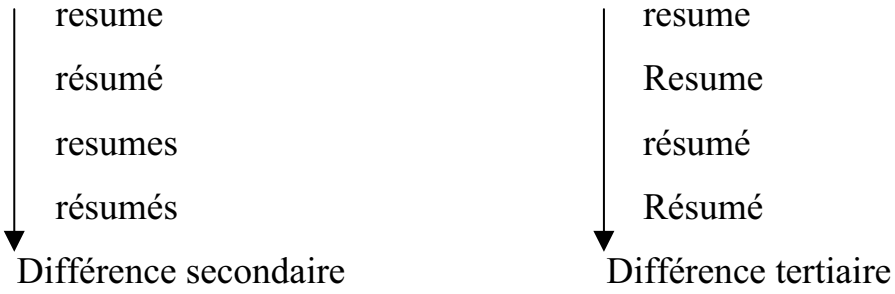
Niveaux de comparaison

On traite ces problèmes à l'aide de plusieurs niveaux de comparaison et par l'attribution d'une différence secondaire ou tertiaire aux lettres selon la présence de diacritiques et leur casse (voir la Figure 5-17). Ce qui nous donne les règles suivantes :

R1 Considérer les différences secondaires — uniquement s'il n'y a pas de différence primaire.

R2 Considérer les différences tertiaires — uniquement s'il n'y a ni différence primaire ni secondaire.

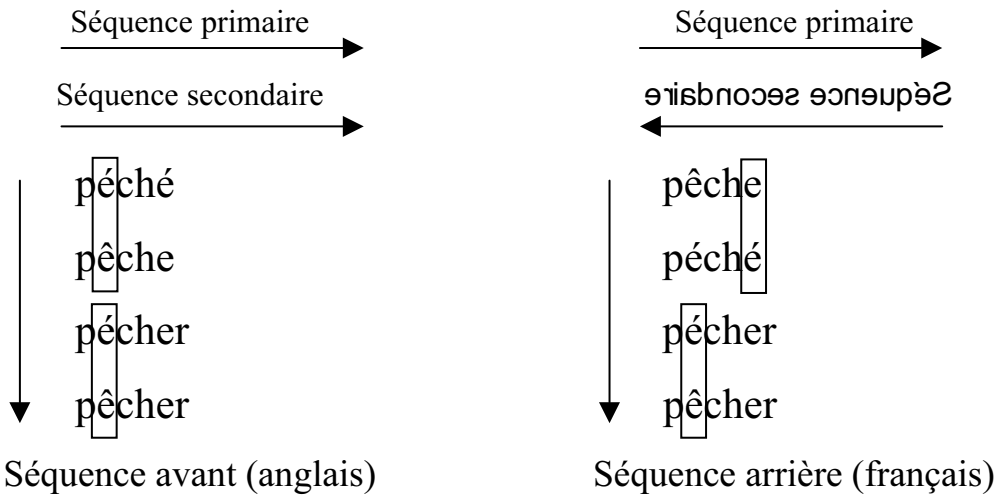
Figure 5.17. Niveaux de comparaison



En anglais et quelques autres langues similaires, les accents n'ont qu'une importance secondaire, la chasse, elle, n'ayant qu'une importance tertiaire. On considère les caractères ignorables (voir ci-dessous) comme des différences secondaires ou tertiaires. Dans d'autres langues et écritures, on considère d'autres traits graphiques comme des différences secondaires ou tertiaires.

Le français présente un cas particulier. En effet, dans le tri lexicographique français, les derniers accents ont plus d'importance que les premiers alors qu'en anglais c'est le contraire. Les cinq premiers caractères de base des deux paires de mots de la *Figure 5-18* sont identiques. Les accents les plus importants sont encadrés.

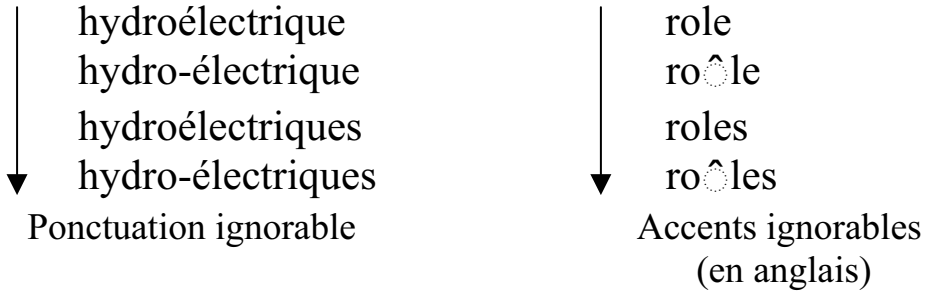
Figure 5.18. Orientations



Caractères « ignorables » ou négligeables

Il existe une autre classe de caractères intéressante, il s'agit des caractères négligeables, ou mieux « ignorables », comme les espaces, les tirets et quelques autres signes de ponctuation. On considère ces caractères quand il n'existe aucune différence plus importante dans la chaîne.

Figure 5.19. Caractères ignorables



La règle générale s'énonce de la façon suivante :

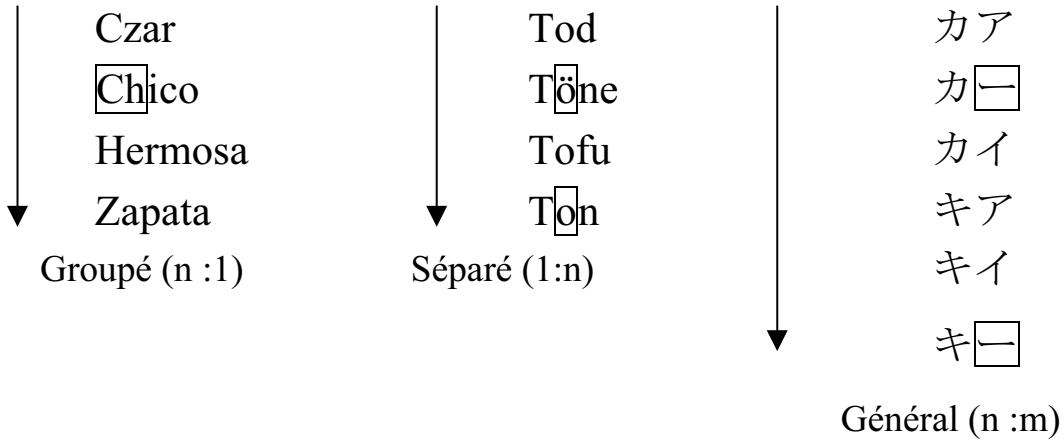
R3 Considérer que les caractères ignorables n'introduisent pas une différence primaire.

Correspondances multiples

Pour plusieurs langues, comme c'est le cas pour les tris traditionnels allemand et espagnol, un caractère peut être considéré dans les comparaisons comme s'il correspondait en fait à deux caractères, ou, inversement, deux caractères peuvent être considérés comme un seul caractère (voir la *Figure 5-20*). Dans le tri lexicographique espagnol traditionnel, par exemple,

« Ch » se classe comme s'il s'agissait d'une seule lettre après Cz et non avant Ci. Dans l'ordre traditionnel allemand, le « ö » se trie comme s'il s'agissait d'un « oe » et se classe donc entre « od » et « of ».

Figure 5.20. Correspondances multiples



En japonais, une marque de longueur ー allonge la voyelle du caractère précédent. Selon la voyelle, le résultat trié sera différent. Ainsi, après le caractère カ « ka », la marque de longueur ー indique un long a et se trie après ア « a » ; alors qu'après le caractère キ « ki », la marque de longueur indique un long i et se classe après イ « i ». On a donc dans l'ordre ka, kâ, kai, kia, kii, kî.

Examinons le second caractère de chaque mot dans la troisième colonne de la *Figure 5-20*. Trois caractères différents se placent en seconde position : ア « a », marque de longueur ー et イ « i ». La règle générale s'énonce de la façon suivante :

R4 Il se peut que l'on doive comparer *N* caractères comme s'ils en formaient *M*.

Tri de caractères non pertinents

Le classement trie les caractères selon les règles en vigueur pour la langue de cet utilisateur. Il est fort possible qu'une liste de termes codés à l'aide du standard Unicode proviennent de plusieurs langues. Ces termes sont tous triés dans l'ordre habituel de la langue de l'utilisateur.

Pour des caractères et des écritures qui ne s'utilisent pas habituellement dans une langue particulière, il se peut qu'il n'existe pas de règles explicites. Ainsi, le français et le suédois peuvent-ils avoir des règles très précises mais différentes de trier un « ä » (après le *z* ou comme un caractère accentué possédant une différence secondaire avec *a*) mais aucun des deux n'a défini un ordre particulier pour les idéogrammes han. Les mises en œuvre d'Unicode fournissent donc d'ordinaire une mise en ordre implicite (à la façon du tri culturellement neutre utilisé dans cet ouvrage pour les idéogrammes). Les lettres latines majuscules et minuscules resteront de la sorte proches pour un utilisateur japonais. L'ordre de classement relatif des différentes écritures est d'ordinaire configurable.

R5 Définir un ordre implicite ou culturellement neutre pour les caractères non pertinents.

Caractères sans correspondance

Une autre solution consiste à traiter les caractères non pertinents de façon encore plus radicale. On retrouve parmi ces caractères des filets, des sortes diverses, des vignettes et parfois des caractères alphabetiques qui n'intéressent pas la clientèle d'une mise en œuvre particulière. En règle générale, on n'attribue pas de poids aux caractères non pertinents pour un tri lexicographique particulier. On réduit de la sorte la taille de la suite de tri. Toutefois, afin de définir un ordre de tri bien défini et efficace, on peut utiliser pour ces caractères leur numéro de caractère Unicode comme poids de tri.

R6 Trier, à une position paramétrisable, les caractères non pertinents dans l'ordre binaire Unicode.

Paramétrisation

Une utilisation efficace du tri implique un certain niveau de paramétrisation de la part des programmeurs. Ceux-ci doivent également pouvoir obtenir suffisamment d'informations sur le résultat du tri. Une des informations nécessaires est la position dans la chaîne où apparaît une différence afin de pouvoir trouver la chaîne initiale commune. Remarquez que cette position n'est pas nécessairement égale dans les deux chaînes. En effet, les équivalences de codage peuvent faire en sorte que les trois premiers caractères d'une chaîne correspondent aux quatre premiers d'une autre.

Paramètres d'entrée

- Règles de la langue active ;
- Ordre relatif des écritures ;
- Précision du tri.

Paramètres de sortie

- Première position dans chaque chaîne où elles diffèrent ;
- Direction et la précision de la différence.

Paramètres de traitement

- Prétraiter chaque chaîne afin d'accélérer la comparaison ;
- Ne traiter que ce qui est nécessaire pour une comparaison autonome

Optimisations

Les comparaisons multiniveaux nécessitent un peu plus de travail que les comparaisons binaires. Bien que des études concrètes estiment l'accroissement de charge à 50 pour cents, il est souvent rentable de transformer en premier lieu les termes à comparer en des *clés de tri* équivalentes. Clés qui lorsqu'elles sont triées fournissent le même résultat qu'un tri binaire des termes originels. (Dans la bibliothèque standard C, la fonction `wcsxfrm` effectue une telle transformation.) Ces clés de tri peuvent être formées d'une chaîne des poids de base, suivie de chaînes représentant les poids utilisés pour les différences secondaires et tertiaires. Contrairement aux valeurs de code d'Unicode les clés de tri n'ont pas besoin d'être codées sur

16 bits[9]. On peut donc utiliser des fonctions hautement optimisées de la bibliothèque standard C comme `strcmp`.

On peut également stocker les clés de tri et éviter de la sorte un calcul inutile quand une liste doit être retriée. Une *comparaison au besoin* constitue une autre forme d'optimisation simple. Pour chaque chaîne, on met alors fin à l'assemblage des clés de tri dès qu'on repère une différence. Cette méthode réduit le calcul nécessaire lorsqu'une différence se retrouve au début de la chaîne.

Repérage

Le repérage et la comparaison partagent de nombreux points communs, parmi lesquels le choix d'une correspondance faible, forte ou exacte. On ajoute souvent d'autres fonctionnalités comme le seul repérage des mots (en d'autres mots, une frontière de mot doit apparaître des deux côtés de la chaîne recherchée). Une technique consiste à coder un tri rapide pour la correspondance faible. Quand on rencontre une chaîne candidate, on vérifie ensuite d'autres critères (correspondance de diacritiques, casse, etc.).

Quand on recherche une chaîne, il faut repérer les caractères de queue sans chasse qui pourraient modifier l'interprétation du dernier caractère à comparer. Ainsi, il se peut que lors de la recherche de « San Jose » on trouve la chaîne candidate suivante « Visiter San José, capitale du Costa Rica… ». Si on effectue une recherche à correspondance exacte (diacritiques importants), il faut alors rejeter cette chaîne. Par contre, pour une correspondance faible, cette chaîne est valable. Il faut inclure tout caractère de queue sans chasse quand on renvoie la position et la longueur de la sous-chaîne repérée. Pour ce faire, on peut utiliser les mécanismes présentés à la *Section 5.15*, *Repérage des frontières d'élément textuel*.

Une des principales utilisations de l'équivalence faible est le repérage indépendant de la casse. Beaucoup de mises en œuvre traditionnelles capitalisent la chaîne à rechercher et le texte cible. Malheureusement, les correspondances de casse dépendent de la langue et *ne* sont *pas* univoques (voir les sections *4.1*, *Casse — normatif*. et *7.1, Latin*). On recommande de mettre en œuvre l'indépendance à la casse grâce aux mêmes mécanismes et tableaux que ceux décrits au début de cette section. Il est conseillé, notamment, à de nombreuses applications (par exemple les systèmes de fichier) de traiter certains caractères (I, i, İ *i majuscule point en chef*, ı *i sans point*) comme une seule classe d'équivalence afin de garantir pour le turc, en l'occurrence, des résultats raisonnables.

La correspondance inexacte vers un jeu de caractères externes cause un problème voisin. Afin de régler cette difficulté, il est préférable de regrouper les caractères prêtant à confusion en une classe d'équivalence faible (đ *d barré*, ð *eth*, Đ *d barré majuscule*, Ð *eth majuscule*). Cette méthode correspond généralement mieux à ce que l'utilisateur s'attend à obtenir lorsqu'il recherche des fichiers ou d'autres objets nommés.

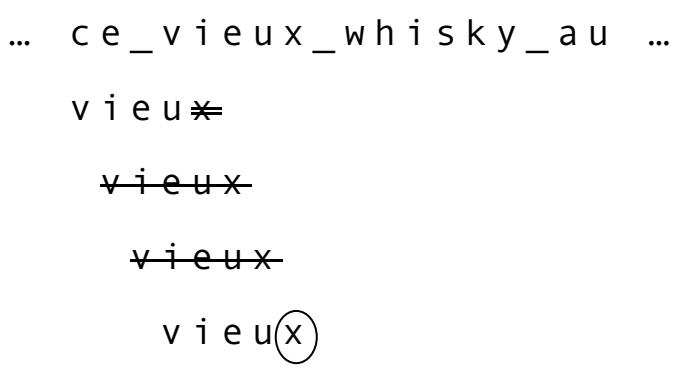
Repérage sublinéaire

Il est clair qu'on peut effectuer une recherche multilingue grâce à l'information de classement, si ce n'est par la force brute. Cependant cette méthode emploie un algorithme de complexité *O(m*n)* dans le pire des cas et *O(m)* dans les cas les plus fréquents, où *n* représente le nombre de caractères à rechercher et *m* le nombre de caractères dans le texte cible.

[9] Cette restriction ne s'applique plus, les numéros de caractères Unicode sont abstraits, les unités de stockage pour leur part peuvent prendre différentes largeurs, y compris 32 bits.

Il existe une série d'algorithmes publiés qui permet de repérer un texte simple, tout en utilisant une méthode sublinéaire. Ces algorithmes parviennent à une complexité *O(m/n)* dans les cas les plus fréquents en ignorant certains caractères dans le texte cible. Plusieurs mises en œuvre ont adapté un de ces algorithmes afin qu'il effectue sa recherche sur des textes prétraités conformément à l'algorithme de tri, on obtient alors une recherche rapide pour des textes dans la langue prétraitée (voir la *Figure 5-21*).

Figure 5-21. Repérage sublinéaire

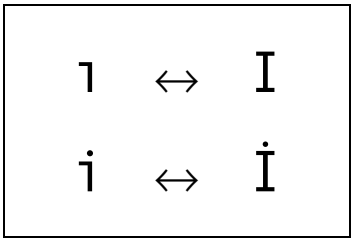


Les principaux problèmes d'adaptation d'un algorithme de tri sensible à la langue, pour en faire un algorithme sublinéaire, ont trait aux correspondances multiples et aux caractères ignorables. Les algorithmes sublinéaires précalculent les tableaux d'information. Les tableaux multiniveaux présentés à la *Figure 5-1* sont des mécanismes efficaces de réduction de l'espace mémoire.

5.18 Correspondance de casses

L'immense majorité des correspondances de casse est uniforme, toutes langues confondues. Il arrive, bien que rarement, que la correspondance entre les formes majuscules et minuscules d'une même lettre diffère pour deux langues qui emploient le même système d'écriture. L'exemple le plus connu est sans doute le turc, où U+0131 ı LETTRE MINUSCULE LATINE I SANS POINT a comme majuscule U+0049 I LETTRE MAJUSCULE LATINE I, alors que U+0069 i LETTRE MINUSCULE LATINE I correspond à U+0130 İ LETTRE MAJUSCULE LATINE I POINT EN CHEF. Voir la *Figure 5-22*.

Figure 5-22. Transformation de casse du i turc



La transformation de casse connaît d'importantes exceptions. Veuillez vous référer au fichier SpecialCasing.html, sur le disque optique, pour plus de renseignements sur ces exceptions. Pour plus d'informations sur la correspondance de casse, voyez la *Section 4.1*, *Casse — normatif*, et le rapport technique Unicode n° 21 *Case Mappings* sur le disque ou, pour une version tenue à jour, sur le site Internet d'Unicode.

Transformation de casse non réversible. Il est important de souligner que les transformations de casse ne sont pas toujours réversibles. De plus, étant donné que de nombreux caractères sont *unicaméraux*[10] *(*comme la plupart du bloc API, par exemple), capitaliser une chaîne ne signifie pas qu'elle ne contiendra plus de caractères minuscules.

Le résultat d'un changement de casse peut avoir une longueur différente de la chaîne d'origine. Ainsi, U+00DF ß LETTRE MINUSCULE LATINE S DUR devient-elle « SS » en majuscule.

Comme nous le verrons à la section *7.2*, *Grec*, on peut également considérer que l'iota souscrit utilisé dans les textes anciens a une correspondance de casse spéciale. Habituellement, les formes majuscules et minuscules d'alpha-iota-souscrit sont réversibles. Parfois, quand les mots prennent leur forme archaïque, on supprime les diacritiques et transforme l'iota-souscrit en un iota majuscule (on peut même supprimer les espaces).

Remarquons que la transformation de casse d'un texte n'est pas réversible. Par exemple :

majuscules(minuscules("Jean Dupont")) ⟶ "JEAN DUPONT".
minuscules(majuscules("Jean Dupont")) ⟶ "jean dupont".

Il existe même des mots simples comme *verderLa* en italien, *prie-Dieu* en français et *McGowan* en anglais qui ne sont ni écrit en majuscules, ni en minuscules, ni en casse de titre. On appelle parfois cette casse mixte, *intracapitale*. Certains caractères n'ont pas de transformation réversible. Ainsi, U+03C2 ς LETTRE MINUSCULE GRECQUE SIGMA FINAL devient en majuscule U+03A3 Σ LETTRE MAJUSCULE GRECQUE SIGMA, mais ce sigma majuscule a pour forme minuscule U+03C3 σ LETTRE MINUSCULE GRECQUE SIGMA, une forme non finale.

On recommande que les texteurs qui utilisent des séquences de touches de commande uniques pour sélectionner la casse conservent la chaîne d'origine et la renvoient avec la suite de touches. En réponse à une série de touches de commande, l'interface-utilisateur produira alors les résultats suivants. Remarquez que la chaîne d'origine réapparaît toutes les quatre fois.

1. Portez ce vieux whisky[11]
2. PORTEZ CE VIEUX WHISKY
3. portez ce vieux whisky
4. Portez Ce Vieux Whisky
5. Portez ce vieux whisky

Un texteur est libre de représenter la casse (majuscule, minuscule, casse de titre) à l'aide d'un style de caractère. L'élimination du style de caractère révèle le texte original. Toutefois, si on adopte cette technique, il faut alors que les correcteurs orthographiques aient accès au style de la casse afin de pouvoir vérifier l'orthographe en fonction de l'apparence réelle du texte.

Pour plus de renseignements sur les conversions de casse, la détection de casse et la comparaison indépendante de la casse, veuillez consulter le rapport technique Unicode n° 21, *Case mappings* sur le disque ou, pour une version tenue à jour, sur le site Internet Unicode.

[10] Ne connaissent pas de distinction de casse.

[11] Extrait du célèbre pangramme Portez ce vieux whisky au juge blond qui fume.