Une extension de TeX incluant Unicode et des filtres de type Lex

Yannis Haralambous, John Plaice

Cahiers GUTenberg

UNE EXTENSION DE TEX INCLUANT UNICODE ET DES FILTRES DU TYPE LEX

Yannis HARALAMBOUS, John PLAICE

Cahiers GUTenberg, nº 20 (1995), p. 55-79.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1995___20_55_0>

Ω, une extension de TEX incluant UNICODE et des filtres de type Lex

Yannis HARALAMBOUS[a] et John PLAICE[b]
Traduction française d'Eric PICHERAL (CRI Rennes)

[a] *187, rue Nationale*
59800 Lille, France
haralambous@univ-lille1.fr

[b] *Département d'Informatique*
Université Laval, Ste-Foy
Québec, Canada G1K 7P4
plaice@ift.ulaval.ca

Résumé. Ω comprend un certain nombre d'extensions de TEX qui améliorent ses possibilités de traitement multilingue. On peut avoir plusieurs jeux de caractères en entrée comme en sortie, et un nombre quelconque de codages internes. On peut définir des automates à états finis en utilisant une syntaxe à la `flex` pour passer d'un codage à l'autre.

Dans cet article, on présente à la fois une introduction technique et quelques applications traitées par l'implémentation actuelle de Ω. Ces applications concernent des problèmes de composition que TEX (et par conséquent aucun autre système connu des auteurs) ne sait résoudre. Elles sont variées, depuis le traitement des fontes calligraphiques (Adobe Poetica) jusqu'à la simple composition du néarlandais, du portugais ou du turc, en passant par l'arabe avec ses voyelles, ou encore un crénage correct du cambodgien.

On mentionne quelques problèmes que Ω ne sait pas résoudre et qui sont des défis pour de futures versions de Ω.

Abstract. *Ω consists of a series of extensions to TEX that improve its multilingual capabilities. It allows multiple input and output character sets, and will allow any number of internal encodings. Finite state automata can be defined, using a* `flex`*-like syntax, to pass from one coding to another.*

In this paper both a technical introduction and a few applications of the current implementation of Ω are given. The applications concern typesetting problems that cannot be solved by TEX (consequently, by no other typesetting system known to the authors). They cover a wide range, going from calligraphic script fonts (Adobe Poetica), to plain Dutch/Portuguese/Turkish typesetting, to vowelized Arabic, fully diacriticized scholarly Greek, or decently kerned Khmer.

A few problems Ω cannot *solve are mentioned, as challenges for future Ω versions.*

Au départ, l'idée de base de Ω consiste à séparer les trois principaux composants d'un système de composition :

Interface utilisateur — Échange d'information — Typographie

Ces composants ne sont pas clairement séparés dans TEX, et cela crée de nombreux problèmes quand il s'agit de traiter d'autres langues que l'anglais. Voici un exemple de

cette situation, signalée dans [14] : pour composer le caractère cyrillique ц en utilisant les macros et les polices de l'AMS (Société Mathématique Américaine) [1], il faut saisir les codes ASCII `ts`. La saisie de `ts` fait partie de l'interface utilisateur de TEX. On obtient ц grâce au composant typographique de TEX. On néglige le composant échange d'information : un texte codé en KOI-8 ou en ISO 8859-5 ne peut être traité directement en utilisant ce schéma : il faut utiliser un pré-traitement ou bien des caractères actifs pour la translitération AMS. Mais l'algorithme de césure — qui dans le cas de l'anglais décomposera la ligature « fi » en « f-i » — peut séparer ц en `т-с`, ce qui est inacceptable. Cela survient parce que la césure (qui est un processus appartenant clairement au composant typographique de TEX) est activée alors que les données sont encore codées de la même façon que dans l'interface utilisateur : ц n'est pas encore reconnu comme une entité indivisible, TEX ne connaît que la paire de codes ASCII `ts`.

Avec Ω, on évite ce problème ; de plus, on peut utiliser n'importe quel codage d'échange pour saisir le texte, ce qui est possible parce que des processus séparés convertissent les flux d'information entre les trois composants : dans l'interface utilisateur, on peut très bien taper les caractères `ts` qui seront convertis en un caractère ISO 10646/UNICODE de code `0x0446` (CYRILLIC SMALL LETTER TSE). Un texte codé selon un codage d'échange quelconque, qu'il soit 8 ou 16 bits, sera aussi converti en ISO 10646/UNICODE. Enfin, avant de passer la main au composant typographique de Ω, les caractères ISO 10646/UNICODE seront traduits selon les positions correspondantes des polices de sortie, et alors seulement Ω activera l'algorithme de césure.

Cet article comprend deux parties : la première, qui émane de [13], est une introduction technique à Ω ; la deuxième, provenant de [8], décrit quelques applications simples, principalement dans le domaine de la composition multilingue.

1. Une introduction à Ω

1.1. Codages et recodages

Si on fait abstraction des problèmes liés à la mise en page, la composition peut être perçue comme un processus qui convertit une chaîne de caractères en une chaîne de glyphes. Ce processus peut être très simple ou très complexe. Le cas le plus simple est sans doute l'anglais où, dans la plupart des cas, le codage d'entrée comme le codage des polices en sortie est l'ASCII ; aucune conversion n'est ici nécessaire. À l'opposé, on pourrait imaginer une transcription latine de l'arabe qui produise un texte en arabe avec toutes ses voyelles et ligatures ; ici il faudra interpréter la translitération, puis déterminer le tracé correct de chaque consonne, et enfin choisir les ligatures et placer les voyelles — le processus est bien plus complexe.

TEX suppose qu'il y a deux codages de base : le codage d'entrée et le codage interne, chacun d'entre eux utilisant un maximum de 8 bits. La conversion du codage d'entrée vers le codage interne se fait par le biais du tableau `xord`. Un caractère en entrée est lu et converti selon le tableau `xord`. Le codage de sortie est le même que le codage

interne, à la différence près que plusieurs caractères peuvent se combiner pour former une ligature.

Supposons que l'on travaille dans un environnement hétérogène et que l'on reçoive régulièrement des fichiers utilisant des codages différents. Dans ce cas, on se heurte à un problème, car la conversion du codage d'entrée vers le codage interne est faite en dur dans le code de TEX. Pour changer le codage d'entrée, il faut modifier le code de TEX, ce qui est difficilement acceptable.

Alors comment contourner ce problème ? La première possibilité consiste à utiliser un préprocesseur, avant d'appeler TEX. On peut aussi utiliser des caractères actifs. Au début de chaque fichier, certains caractères sont définis comme étant des macros. Mais cette méthode est peu fiable, car d'autres macros peuvent considérer que ces caractères sont des lettres ordinaires.

Il serait bien plus intéressant d'avoir une commande déclarant que le codage d'entrée a été modifié, et de permettre à TEX de changer de processus de conversion à la volée, en gardant le même codage interne.

Il serait sans doute relativement facile d'adapter TEX pour qu'il puisse rapidement basculer d'un codage de caractères sur un octet à un autre. Mais il existe aujourd'hui plusieurs jeux de caractères multi-octets : JIS, Shift-JIS et EUC au Japon, GB en Chine et KSC en Corée. Certains sont de longueur fixe, d'autres ont des codes à état de longueur variable. Le plan de base de ISO 10646-1.2 (Unicode-1.1) étant maintenant défini, on dispose d'un jeu de caractères 16 bits qui peut être utilisé pour traiter la plupart des langues dans le monde. Pourtant, pour des raisons de compatibilité, on rencontre souvent des fichiers en format UTF, dans lesquels on peut stocker jusqu'à 32 bits avec un codage de longueur variable (1-6 octets), mais les octets ASCII y restent des octets ASCII. En d'autres termes, le processus de conversion du codage d'entrée vers le codage interne n'est pas vraiment simple.

Pour compliquer encore les choses, la nature du codage interne n'est pas évidente du tout. Doit-il être fixe, auquel cas la seule solution raisonnable est l'ISO 10646/UNICODE, ou bien variable ? Si le codage interne est fixe, dans la plupart des cas, une conversion du codage interne vers le codage police devra aussi être effectuée. Certaines polices japonaises sont par exemple codées en interne suivant l'unification Han, le principe qui sous-tend ISO 10646/UNICODE. Le codage interne devrait plutôt suivre les nombres de Kuten ou l'un des codages du JIS. Si c'est la même chose pour le codage d'entrée, une double conversion, pas toujours simple ni nécessaire, doit être alors réalisée.

Pour rendre l'affaire encore plus difficile, l'éditeur peut très bien ne pas disposer des fontes nécessaires pour une langue particulière. Une translitération est alors indispensable, mais elle est complètement indépendante du codage des caractères ; après tout, on peut utiliser la même translitération de caractères latins vers le cyrillique, que l'on utilise ISO 646 ou ISO 10646/UNICODE. Et la translitération n'a rien à voir avec le codage des polices. En fait, on voudrait utiliser les mêmes fontes arabes, qu'on effectue la saisie en utilisant une translitération latine en ISO 8859-1, ou arabe en ISO 8859-6 ou ISO 10646/UNICODE. En plus, l'ordre des caractères dans un flux d'entrée peut ne pas correspondre à l'ordre dans lequel les caractères doivent être placés sur le papier ou l'écran.

Par exemple, [4], de nombreuses voyelles cambodgiennes sont scindées en deux : une partie est placée à gauche d'un groupe consonantique, l'autre à droite. Des problèmes similaires ont surgi avec le cingalais et les autres écritures du sous-continent indien [7].

Finalement, on doit réaliser que des messages d'erreur et les traces doivent être affichés et qu'ils n'utiliseront pas forcément le même jeu de caractères que celui du codage d'entrée ou du codage interne.

1.2. Translitération et analyse contextuelle

Il semble clair que le seul codage interne viable soit le codage police. Pourtant, il n'y a aucune raison pour que la conversion du codage d'entrée vers le codage interne se fasse en une seule étape. De toute évidence, on peut toujours l'effectuer et, si les polices sont assez grandes, on peut toujours faire toute l'analyse au niveau ligature dans la police. Mais ce choix nous empêche de séparer des tâches distinctes comme — par exemple pour l'arabe — la conversion initiale de tout le texte en ISO 10646/UNICODE, puis la translitération, ensuite le calcul de la forme appropriée de chaque lettre, et seulement alors l'activation du mécanisme de ligature des polices.

En fait, nous proposons d'autoriser l'écriture d'un nombre quelconque de filtres, et la possibilité que la sortie d'un filtre devienne l'entrée d'un autre filtre, comme pour les *pipes* d'Unix.

1.3. Processus de translation de Ω

Dans Ω, ces filtres sont appelés Processus de Translation de Ω (ΩTPs). Chaque ΩTP est défini par l'utilisateur dans un fichier `.otp` : avec une syntaxe rappelant celle du générateur d'analyseur lexical `flex`, les utilisateurs peuvent définir des automates à états finis de Mealy pour transformer les flux de caractères en de nouveaux flux de caractères.

Ces tables de traduction utilisateur ne sont pas directement lues par Ω. Mais des représentations plus compactes (fichiers `.ctp`) sont produites par le programme `otptoctp`. La lecture d'un fichier `.ctp` se fait par l'intermédiaire d'une primitive Ω (voir ci-dessous).

Voici la syntaxe du fichier de translation :

`in:`	*n*;
`out:`	*n*;
`tables:`	T^*
`states:`	S^*
`aliases:`	A^*
`expressions:`	E^*

où *n* représente un nombre quelconque. Les nombres peuvent être représentés en notation décimale, octale à la `WEB` (`@'`...) ou hexadécimale (`@"`...), ou encore être des caractères ISO 646 affichables inclus entre un accent grave et une apostrophe.

Le premier (resp. deuxième) nombre spécifie le nombre d'octets dans un caractère en entrée (resp. sortie), le défaut étant de un dans les deux cas. Ces valeurs sont nécessaires pour préciser quel processus de translation utiliser pour faire la conversion depuis ou vers un jeu de caractères qui utilise plus d'un octet par caractère.

Ces tables sont régulièrement utilisées dans les conversions de jeu de caractères, quand on ne peut pas spécifier facilement des approches algorithmiques. Voici la syntaxe d'une table *T* :
id [*n*] = {*n, n, . . . , n*} ;

Les ΩTP, comme dans `flex`, autorisent un nombre donné d'états. Chaque expression est valide seulement dans un état donné. L'utilisateur peut spécifier des changements d'états. Les états sont souvent utilisés pour l'analyse contextuelle. La syntaxe pour un ensemble *S* d'états est :
id, *id*, . . . , *id* ;

Les expressions sont des paires motif-action. Les motifs sont écrits comme des expressions régulières simples, et on peut en définir des alias. La syntaxe d'un alias *A* est :
id = *L* ;
où *L* est un motif.

Si on utilise un seul état, alors une expression *E* est composée d'un motif et d'une action :
L => *R** ;
et la syntaxe d'un motif est :

L	::=	*n*	
	|	*n*–*n*	intervalle
	|	.	joker
	|	*LL*	concaténation
	|	*L*{*n, m*}	occurrences
	|	(*L* | . . . | *L*)	choix
	|	^ (*L* | . . . | *L*)	choix négatif
	|	{*id*}	abréviation ;

alors que la syntaxe simplifiée pour une action est :

R	::=	*string*	
	|	*n*	
	|	*n*	
	|	\($ − *n*)	
	|	\(* + *n* − *n*)	
	|	#(*R*)	
	|	*id* [*R*]	
	|	*R op R*	arithmétique ;

Les motifs sont appliqués à l'entrée. Quand il y a correspondance entre un motif et le flux d'entrée, l'action en partie droite est exécutée. Une *chaîne* est simplement émise en sortie. Le *n* correspond au *n*e caractère correspondant et le \$ correspond au dernier caractère apparié. * correspond à toute la chaîne appariée et \(* − *n*) correspond

à l'ensemble de la chaîne moins les *n* derniers caractères. La recherche dans la table se fait en utilisant des crochets. Toutes les expressions doivent être précédées d'un #.

Voici un exemple de translation du codage chinois GB 2312-80 vers ISO 10646/UNICODE :

```
in:  1;
out: 2;
tables: tabgb[8795] = {...};
expressions:
(@"00-@"A0)              => \1;
(@"A1-@"FF)(@"A1-@"FF)  =>
     #(tabgb[(\1-@"A0)*@"64 + (\2-@"A0)]);
..                       => @"FFFD;
```

où nous utilisons `0xfffd` (**REPLACEMENT CHARACTER**) comme caractère d'erreur.

Et voici une translitération courante dans les écritures indiennes :

```
{consonante}{1,6} {voyelle}  => \$ \(*-1);
```

La voyelle en fin d'expression est mise avant le flux de consonnes.

La syntaxe complète des expressions est plus compliquée, car il y peut y avoir plusieurs états de traitements. En outre, on peut empiler des valeurs sur la pile d'entrée. Voici la syntaxe complète :
<état> L => R <= R* <nouvel_état>*
état signifie que si l'ΩTP est dans cet état, alors cette paire motif-action peut être utilisée. *nouvel_état* désigne le nouvel état si cette paire motif-action est choisie.

Et voici un exemple d'analyse contextuelle de l'arabe :

```
<MEDIAL>{QUADRIFORM}{NOT_ARABIC_OR_UNI}
        => #(\1 + @"DD00)
        <= \2
        <pop:>
        ;
```

Quand on est dans l'état `MEDIAL` (au milieu d'un mot) et qu'une lettre qui a quatre formes possibles est suivie d'une lettre non arabe, on obtient en sortie la lettre à quatre formes plus la valeur `@"DD00`. La lettre non arabe est replacée sur la pile d'entrée. L'état courant est alors dépilé et l'ΩTP revient à l'état précédent, quel qu'il soit.

1.4. Chargement des ΩTP

Le chargement d'un ΩTP est similaire à celui d'une police. L'instruction est simplement :

\otp*nouveau_nom* = *nom_de_fichier*
Le fichier .ctp *nouveau_nom*.ctp est lu et stocké dans la mémoire otp info, similaire à la mémoire font info. Une valeur est affectée à la séquence de commande *nouveau_nom*, comme pour les fontes. On peut ensuite faire référence à cet ΩTP par l'intermédiaire de cette valeur ou bien grâce à la commande nouvellement définie.

1.5. Codage d'entrée

À la lecture d'un fichier d'origine inconnue, utilisant un jeu de caractères inconnu, il faut trouver un mécanisme pour déterminer le jeu de caractères. Il y a deux possibilités. On peut utiliser un jeu par défaut, ou bien trouver un moyen de reconnaître rapidement de quel jeu il s'agit.

Heureusement, l'ISO 646 est un sous-ensemble de la plupart des jeux de caractères. Le jeu ISO 10646/UNICODE, dans ses versions 16 et 32 bits, a le même codage que ISO 646 pour les 128 premiers caractères. Le seul jeu largement répandu qui ne réponde pas à ce critère est l'EBCDIC d'IBM.

Aussi fournissons-nous le moyen de détecter automatiquement la famille du jeu de caractères. Il suffit que l'utilisateur place un commentaire au tout début de chaque fichier : le caractère % est suffisant pour distinguer chacune de ces familles. Un fichier qui utilise une extension 8 bits de l'ISO 646 commence avec le code caractère 0x25 ; un fichier avec des caractères 16 bits commence par 0x00 0x25[1]. Enfin, un fichier utilisant le codage EBCDIC commence par 0x6c.

S'il n'y a pas de caractère %, alors on suppose qu'il s'agit du codage d'entrée par défaut (ISO 646).

Une fois que Ω sait comment lire les lettres de base latines, il est possible de *déclarer* quelle traduction l'entrée doit subir. Ceci est fait en utilisant la commande
\InputTranslation
Par exemple,
\InputTranslation 1
déclare que tout le flux d'entrée, commençant immédiatement *après* le caractère « retour chariot » à la fin de cette ligne, devra être traité par le processus ΩTP.

Il est aussi possible de changer de jeu de caractères à l'intérieur d'un fichier. Ce processus est plus difficile à réaliser, car il n'est pas toujours évident de savoir *exactement* où la modification doit prendre effet. Supposons que nous passions d'un jeu de caractères 8 bits à un jeu 16 bits. Il est important de savoir quel est le *dernier* caractère 8 bits et quel est le *premier* caractère 16 bits.

On peut résoudre cette question en choisissant un caractère particulier pour marquer le changement. Pourtant, pour simplifier les choses, nous supposons que toutes les modifications de traduction en entrée ont lieu *immédiatement après* le caractère « retour chariot » à la fin de la ligne dans laquelle \InputTranslation apparaît.

[1] Un fichier avec des caractères 32 bits devrait commencer par 0x00 0x00 0x00 0x25, mais la version actuelle de Ω ne traite pas les caractères 32 bits.

1.6. Translitération

Une fois que les caractères ont été lus, le plus souvent suivant un certain jeu de caractères universel comme ISO 10646/UNICODE, l'analyse contextuelle peut avoir lieu, indépendamment du jeu de caractères de départ. Cette analyse peut nécessiter plusieurs filtres, chacun d'entre eux étant similaire au processus de traduction auquel l'entrée a été soumise.

Puisque le nombre de filtres que nous pouvons vouloir utiliser est arbitrairement grand, il y a deux commandes pour spécifier des filtres :
`\NumberInputFilters` *n*
déclare que les *n* premiers filtres sont actifs. La sortie du i^{e} filtre devient l'entrée pour le $i + 1^{\text{e}}$ filtre, pour $i < n$.
`\InputFilter` *m i*
déclare que le m^{e} filtre est le i^{e} ΩTP.

Les séquences de caractères composées successivement des codes caractères 5, 10, 11 et 12 traversent les *n* processus de traduction. Cela signifie que le résultat du dernier processus de traduction sera le codage fonte en sortie ; c'est sur ce codage que sera appliqué l'algorithme de césure.

Notre exemple sur l'arabe se présentera comme suit :

```
\otp\trans     = ISO646toISO10646
\otp\translit  = TeXArabicToUnicode
\otp\fourform  = UnicodeToContUnicode
\otp\genoutput = ContUnicodeToTeXArabicOut
\InputFilter 0 \translit
\InputFilter 1 \fourform
\InputFilter 2 \genoutput
\NumberInputFilters 3
```

Le traducteur `TeXArabicToUnicode` sélectionne la translitération latine et fait la conversion en arabe. Comme pour `UnicodeToContUnicode`, il effectue l'analyse contextuelle pour l'arabe, c'est-à-dire qu'il prend l'arabe en entrée (en ISO 10646/UNICODE) et, en utilisant une zone privée, détermine laquelle des quatre formes (isolée, initiale, médiale ou finale) doit prendre chaque consonne. Finalement, `ContUnicodeToTeXArabic` détermine quel emplacement dans la police correspond à chaque caractère. Évidemment rien n'empêche la police d'avoir aussi son propre mécanisme sophistiqué de ligatures.

1.7. Codages de sortie et codages spéciaux

TEX ne se borne pas à produire des fichiers `.dvi`. Il crée aussi des fichiers `.aux`, `.log` et bien d'autres, qui peuvent à leur tour être relus par TEX. Il est donc important que le mécanisme de sortie soit aussi général que celui d'entrée. Pour cela, nous

introduisons les opérations duales :

```
\OutputTranslation
\OutputFilter
\NumberOutputFilters
```

avec évidemment les arguments appropriés.

De façon similaire, TEX peut écrire dans ses fichiers `.dvi` des commandes spécifiques à un pilote de périphérique, en utilisant des commandes `\special`. Comme les arguments de `\special` sont eux-mêmes des chaînes, il semble judicieux d'autoriser aussi les commandes suivantes :

```
\SpecialTranslation
\SpecialFilter
\NumberSpecialFilters
```

1.8. Fontes de grande taille

TEX limite la taille des fontes à un maximum de 256 caractères. Pourtant, le besoin en fontes de plus grande taille se fait souvent sentir. Un total de 256 caractères est évidement insuffisant pour les langues utilisant des idéogrammes. Mais la même remarque s'applique aux écritures alphabétiques comme le latin, le grec ou le cyrillique ; pour chacune d'entre elles, ISO 10646/UNICODE prévoit plus que 256 caractères. Cependant nombre de ces caractères étant des combinaisons de base « caractère+signe diacritique », le nombre réel de glyphes de base est donc plus réduit. En fait, pour chacun de ces trois alphabets, une seule fonte de 256 caractères suffit pour les glyphes de base.

Nous avons donc décidé, dans une première étape, d'offrir le moyen de traiter des fontes virtuelles de grande taille (16 bits), dont les glyphes de base tiendront en fait dans une fonte à 8 bits. Ceci n'est évidemment qu'une première étape, mais elle présente l'avantage de permettre le traitement de fontes de grande taille, y compris leurs mécanismes de ligature, sans avoir à réecrire tous les pilotes de sortie.

En plus des changements à TEX, il faudra aussi modifier `dvicopy` et `vptovf`, qui deviennent respectivement `xdvicopy` et `xvptovf`. Les fichiers `.tfm`, `.vp` et `.vf` sont remplacés par les fichiers `.xfm`, `.xvp` et `xvf` respectivement. Les nouveaux programmes peuvent bien sûr toujours lire les anciens fichiers.

1.9. Fichiers `.xfm`

Les fichiers `.xfm` sont similaires aux fichiers `.tfm`, à la différence près que la plupart des valeurs utilisent 16 ou 32 bits, et ont donc doublé de taille. L'en-tête comprend 13 mots de quatre octets. Pour distinguer les fichiers `.tfm` et `.xfm`, le premier mot est entièrement mis à zéro. Les douze mots suivants contiennent les valeurs de *lf*, *lh*, *bc*,

ec, *nw*, *nh*, *nd*, *ni*, *nl*, *nk*, *ne* et *np*. Toutes ces valeurs doivent être positives ou nulles et inférieures à 2^{31}. Désormais, chaque valeur de *char_info* est définie comme suit :

width index	16 bits
height index	8 bits
depth index	8 bits
italic index	14 bits
tag	2 bits
remainder	16 bits

Chaque *lig_kern_command* est de la forme :

op byte	16 bits
skip byte	16 bits
next char	16 bits
remainder	16 bits

Finalement ces extensions prennent deux fois plus de place.

1.10. Fichiers `.xvp`

Les fichiers `.xvp` sont simplement des fichiers `.vpl` dans lesquels toutes les restrictions dues au codage des caractères sur 8 bits ont été levées. Sinon, tout le reste est identique.

1.11. Modifications mineures

Comme les changements mentionnés ci-dessus ont nécessité un examen approfondi de tout le code de TEX, nous en avons profité pour supprimer toutes les restrictions dues à un codage sur un seul octet. Par exemple, on peut utiliser maintenat plus de 256 registres (de chaque sorte). De même, on peut avoir plus de 256 fontes simultanément.

2. Applications de Ω

2.1. Néarlandais, portugais, turc : un traitement facile

Ces trois langues (et peut-être d'autres ?) ont au moins une chose en commun : elles ont besoin de fontes ayant une table de ligatures légèrement différente de celle du codage de Cork. La composition du hollandais utilise la fameuse ligature « ij » (qu'on trouve par exemple dans le nom de personnes connues comme Dijkstra, van Herwijnen, Eijkhout, van Dijk, ou encore celui de la ville de Nijmegen ou du lac IJselmeer) ; cette ligature apparaît dans le codage de Cork (comme dans ISO 10646/UNICODE), mais jusqu'à maintenant, il n'existait pas de moyens tranparents à l'utilisateur pour l'obtenir. Avec Ω il suffit de placer une macro appelant un filtre ΩTP spécifique dans l'expansion de la macro qui bascule en hollandais ; en syntaxe Ω (décrite dans la section 1.3), on

peut écrire cet ΩTP très simplement :

```
in: 1
out: 2
expressions:
'I''J' => @"0132;
'i''j' => @"0133;
.      => \1;
```

où 0x0132 et 0x0133 sont les caractères « IJ » (LATIN CAPITAL LIGATURE IJ) et « ij » (LATIN SMALL LIGATURE IJ) en ISO 10646/UNICODE.

Le portugais et le turc n'utilisent pas les ligatures « fi », ... « ffl » (en turc la raison est évidente : l'alphabet turc utilisant les deux lettres « i » et « ı », il serait impossible de savoir si « fi » représente « f » + « i » ou « f » + « ı »). C'est un problème majeur pour TEX, car la seule solution qui permettrait de conserver une saisie naturelle serait d'utiliser une nouvelle fonte ; et définir un ensemble complet de fontes (virtuelle ou réelle), pour éviter seulement 5 ligatures, présente plus d'inconvénients que d'avantages. Ω résoud facilement ce problème ; bien sûr il est impossible de désactiver une ligature, puisqu'elle arrive à la toute dernière étape, à savoir à l'intérieur de la fonte. Nous devons donc tricher d'une façon ou d'une autre ; le plus naturel consiste à placer un caractère invisible entre le « f » et le « i » ; dans ISO 10646/UNICODE, il y a précisément un tel caractère, dont le code est 0x200b (ZERO WIDTH SPACE) ; on peut réaliser cette opération avec une ligne ΩTP du type `'f''i' => "f" @"200b "i"` pour chaque ligature. Ce caractère devra ensuite être mis en correspondance avec le caractère « compound mark » de la table de Cork, qui a été défini pour cet usage.

Une meilleure méthode consisterait à définir un second « f » dans la table de la police de sortie, qui ne formerait pas de ligature avec « f », « i » ou « l ». Cela donnerait la possibilité à la police d'appliquer un crénage entre les deux lettres, et de compenser l'effet de la ligature manquante (après tout, si une police est dessinée pour utiliser une ligature entre « f » et « i », une paire « fi » sans ligature paraîtrait assez étrange et pourrait nécessiter une modification).

2.2. Poetica d'Adobe

Poetica est une famille de polices scriptes chancellières, dessinée par Robert Slimbach et commercialisée par Adobe Systems Inc. D'après la publicité d'Adobe, « *La police Poetica a été conçue d'après les cursives manuscrites diplomatiques produites durant la Renaissance italienne. Élégante et simple, l'écriture diplomatique est reconnue comme étant à la base des polices italiques et elle marque le point de départ de la calligraphie moderne. Robert Slimbach a capté la vitalité et la grâce de ce style d'écriture dans Poetica. Les caractéristiques de l'écriture diplomatique comprennent l'usage fréquent de lettres aux formes fleuries, de ligatures et de caractères variables permettant d'embellir des manuscrits qui sinon auraient un aspect un peu rigide. Pour capter la variété de formes et*

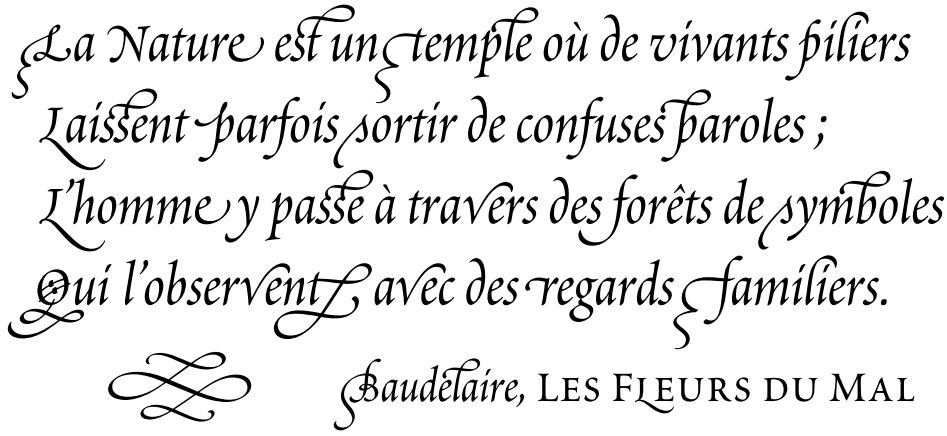


Figure 1 – Exemple du caractère Poetica

la richesse de cette écriture, Slimbach a créé des alphabets supplémentaires et des jeux de caractères en concevant Poetica de main de maître, qui contient un ensemble varié de ces formes de lettres. »

Techniquement, Poetica comprend 21 polices PostScript : Chancery I–IV, Expert, Small Caps, Small Caps Alternate, Lowercase Alternates I–II, Lowercase Beginnings I–II, Lowercase Endings I–II, Ligatures, Swash Caps I–IV, Initial Swash Caps, Ampersands, Ornaments. Alternate, Beginnings, Endings et Ligatures sont particulièrement intéressantes pour nous, car Ω peut automatiquement choisir des caractères dans ces polices. L'utilisateur saisit simplement son texte, en marquant éventuellement le changement d'alphabet par un symbole. Un ΩTP convertit l'entrée en caractères d'une police 16 bits virtuelle, qui contient les caractères de tous les composants de Poetica. En utilisant plusieurs ΩTP et en les modifiant à la volée, l'utilisateur pourra choisir le nombre de ligatures qu'il veut obtenir en sortie. Cela permettra aussi d'aller plus loin qu'Adobe, en définissant le crénage entre des paires de caractères de différentes polices de Poetica.

Voir fig. 1 pour un échantillon de texte composé en Poetica.

2.3. À propos du grec, ancien et moderne (mais ancien plus que moderne)

2.3.1. Signes diacritiques ou crénage ?

Les personnes instruites connaissent en général les lettres grecques. Au collège déjà, ayant utilisé θ pour les angles, γ pour l'accélération et π pour calculer la surface d'un cercle de rayon donné, nous sommes familiarisés avec toutes ces lettres, comme avec l'alphabet latin. Mais pour écrire en grec, en particulier en grec ancien, il ne faut pas seulement des lettres. On utilise deux types de signes diacritiques, les accents (aigu, grave et circonflexe), et les esprits (rude et doux) qui sont placés sur les voyelles et sur la consonne rho.

Chaque mot possède au plus un accent[2], et 99.9 % des mots grecs ont effectivement *un* accent. Chaque mot commençant par une voyelle a exactement *un* esprit[3]. Il y a donc en grec beaucoup plus d'accentuation que dans tout autre langue à alphabet latin, à l'exception évidente du viêtnamien.

Comment TEX traite-t-il les signes diacritiques grecs ? En suivant l'approche traditionnelle comme le fait la primitive `\accent`, nous n'aurions eu pratiquement *aucune* césure (ce qui aurait entraîné de désastreux sur/sous remplissages de ligne, car le grec peut facilement avoir des mots de grande longueur tels que ὠτορινολαρυγγολογικός), *aucun* crénage, et une saisie très malcommode, avec une ou deux macros à chaque mot.

La première approche, dont Silvio Levy est l'auteur [11], consiste à utiliser les ligatures de TEX (les plus « bêtes » d'abord, puis les plus « intelligentes »), pour obtenir les lettres accentuées comme combinaison de codes représentants les esprits (> et <), les accents (‘,’ et =) et les lettres elles-mêmes. De cette façon, on écrit `>‘h` pour obtenir ἡ. Cette approche résolvait le problème de la césure et de la difficulté de la saisie.

Néanmoins, cette approche ne résoud pas le problème du crénage. Prenons le cas fréquent de l'article τὸ (lettre tau suivie de la lettre omicron) ; dans presque toutes les polices, il existe une instruction de crénage entre ces deux lettres, à cause bien sûr des caractéristiques invariantes de leur dessin. Supposons maintenant que omicron soit accentué, et qu'on écrive `t‘o` pour obtenir tau suivi de omicron avec accent grave. Ce que TEX voit est un « t » suivi d'un accent grave. Aucun crénage ne peut être défini entre ces deux caractères, car nous n'avons aucune idée de ce qui peut suivre l'accent grave (un iota par exemple, et en général il n'y a pas de crénage entre tau et iota). Quand la lettre omicron arrive, il est trop tard ; TEX a déjà oublié qu'un tau précédait l'accent grave.

Pour résoudre ce problème on pourrait écrire les signes diacritiques *après* les voyelles (« notation post-positive »). Mais cela entre en contradiction avec les caractéristiques visuelles des signes diacritiques associés aux majuscules, puisqu'ils sont placés à gauche de ces dernières : ῞Εαρ peut difficilement utiliser la translitération `E>’ar`. Et après tout, TEX devrait pouvoir composer le grec correctement, quelle que soit la façon dont sont saisis les lettres et les signes diacritiques.

Ω résoud ce problème en utilisant une suite appropriée de processus de traduction (ΩTP), une notion expliquée en section 1.3 : à titre d'exemple, considérons le mot ἔαρ :

1. Supposons que l'utilisateur désire saisir son texte en ASCII 7 bits ; il tapera `>’ear`, ce qui est déjà de l'ISO-646, de sorte qu'aucune traduction en entrée n'est nécessaire. Il pourrait aussi utiliser un codage entrée comme ISO-8859-7 ou ΕΛΟΤ ; l'utilisateur pourra alors aussi bien taper `>'εαρ` que `>έαρ`.[4]. Le premier ΩTP transformera ces codes vers les codes 16 bits appropriés d'ISO 10646/UNICODE : `0x1f14`

[2] Un accent est parfois déplacé d'un mot vers le précédent : ἄνθρωπός τις au lieu de ἄνθρωπος, τίς, de sorte que, typographiquement, un mot peut avoir plus d'un accent.

[3] Avec une exception : les lettres ρρ sont souvent écrites ῥῥ, quand elles sont à l'intérieur d'un mot : ποῥῥῶ.

[4] La raison de la complication absurde qui force à taper `>'ε` ou `>έ` pour obtenir ἔ est que le codage du « grec moderne » a choisi la facilité en n'utilisant qu'un seul accent, comme si la langue grecque était née en 1982, année de la réforme hâtive de l'orthographe décidée pour des raisons politiques.

pour ἕ (GREEK SMALL LETTER EPSILON WITH PSILI AND OXIA), `0x03b1` pour α (GREEK SMALL LETTER ALPHA) et `0x03c1` pour ρ (GREEK SMALL LETTER RHO).

2. Une fois qu'Ω connaît les caractères qu'il manipule (le codage interne par défaut d'Ω est précisément ISO 10646/UNICODE), il fera la césure en utilisant des motifs 16 bits.

3. Finalement, un ΩTP approprié enverra les codes grecs ISO 10646/UNICODE à une fonte 16 bits virtuelle (voir page 69 la raison pour laquelle nous avons besoin de 16 bits), construite à partir d'une ou de plusieurs fontes 8 bits. Cette fonte contient des instructions de crénage, appliquées d'une manière simple, puisque nous nous occupons de trois codes seulement : <ἕ>, <α> et <ρ>. Aucun code auxiliaire n'interfère plus.

4. `xdvicopy`[5] dé-virtualisera le fichier `dvi` et produira un nouveau fichier `dvi` utilisant exclusivement des polices 8 bits, compatibles avec tout pilote `dvi` digne de ce nom.

En séparant les tâches, la césure devient plus naturelle (pour TEX, il faut utiliser des motifs incluant les codes auxiliaires `'`, `‘`, = etc.). Au passage, on a résolu un problème supplémentaire : les primitives `\lefthyphenmin` et `\righthyphenmin` s'appliquent aux caractères de *catcode* 12. Pour obtenir la césure entre des groupes comportant des codes auxiliaires, nous devons déclarer ces codes comme des « caractères de type lettre ». Par exemple, le mot ἕαρ, qu'on écrit `>'ear` : Pour autoriser la césure, les codes `>` et `'` doivent être considérés comme des lettres (`\lccode` significative) ; mais ceci signifie que pour TEX, ἕαρ a 5 lettres au lieu de 3, et en conséquence, même si nous demandons `\lefthyphenmin=3`, le mot sera encore coupé comme ἕ-αρ !! Ω résoud ce problème en coupant *après* la fin de la traduction (dans ce cas `>'e` ou `>'ε` ou `>έ` → ἕ).

2.4. Dactyles, spondées et fontes 16 bits

Les éditions savantes de textes grecs sont légèrement plus compliquées que les éditions ordinaires[6], l'un des ajouts étant un *troisième* niveau de diacritiques : la longueur des syllabes.

On lit dans [2], p. 254 : « *La poésie grecque a été composée sur un principe entièrement différent de celui employé en anglais. Elle n'était pas construite en disposant les syllabes accentuées en motifs, ni avec un système de rimes. Les poètes grecs utilisaient un certain nombre de métriques différentes qui consistaient toutes en un certain arrangement fixe de* ***syllabes longues et courtes*** *»*. Les syllabes longues et courtes sont marquées par les signes diacritiques *macron* et *brève*. Ces diacritiques sont placées *entre* la lettre et le signe diacritique normal, s'il existe (sauf dans le cas de lettres majuscules,

[5] Dans le nom de ce programme, qui est une version étendue du `dvicopy` de Peter Breitenlohner, « x » signifie « extended » et non pas « X-Window ».

[6] Après tout, les érudits étudient les textes grecs depuis plus de 2000 ans

auquel cas elles sont placées après et au-dessus de la lettre alors que les diacritiques normales sont placées à leur gauche).[7]

Les fameux deux premiers vers de l'Odyssée

᾽Ανδρα μοι ἔννεπε, Μοῦσα πολύτροπον, ὃς μάλα πολλὰ
πλάγχθη ἐπεὶ Τροίης ἱερὸν πτολίεθρον ἔπερσε·

forment des hexamètres. Ils comprennent six pieds : quatre dactyles ou spondées, un dactyle et un spondée ou un trochée (voir à nouveau [2] pour plus de détails). On pourrait écrire le texte sans accents ni esprits pour rendre la métrique plus apparente :

Ἄνδρᾰ μοῐ̈ ἔννἔπἔ, Μοῦσᾰ πὸ̆λῡ̄τρὸ̆πὸ̆ν, ὃς μᾰ́λᾰ πο̄̀λλᾰ́
πλᾰγχθῆ ἔπεῑ Τροῑ̄ης ἱ̔ἔρο̄ν πτὸ̆λῐ̔ἔθρὸ̆ν ἔπε̄ρσἔ

ou décider de mettre tous les types de signes diacritiques :

᾽Ἄνδρᾰ μοῐ̈ ἔννἔπἔ, Μοῦσᾰ πὸ̆λύτρὸ̆πὸ̆ν, ὃς μᾰ́λᾰ πο̄̀λλᾰ́
πλᾰ́γχθῆ ἔπεῖ Τροίη̄ς ἱ̔ἔρὸ̀ν πτὸ̆λῐ̔ἔθρὸ̆ν ἔπε̄ρσἔ

Après avoir dépensé une fortune pour acquérir la machinerie qui se trouve entre le clavier et l'écran, on pourrait s'atttendre à ce que la césure et le crénage entre les lettres restent les mêmes, malgré le nombre toujours croissant de signes diacritiques. En fait, ce n'est pas possible avec TEX : il existe exactement 345 combinaisons possibles entre les lettres grecques et les accents, esprits, longueur de syllabe et iota en indice ; TEX peut gérer un maximum de 256 caractères dans une police. Aussi Ω est-il nécessaire pour la césure de textes grecs, chaque fois que des longueurs de syllabe sont composées.

Dans ce cas, les choses ne se passent pas aussi facilement que dans la section précédente : bien que 345 soit un petit nombre par rapport à 65536 (= 2^{16}), l'ISO a décidé qu'il n'y avait pas assez de place pour toutes les combinaisons d'accents, d'esprits et de longueur de syllabe.[8]

Chaque fois que ISO 10646/UNICODE devient insuffisant pour nos besoins, nous utilisons la *zone privée*. Comme dans la série télé *The Twilight Zone*, tout peut *arriver* dans cette zone privée. Dans le cas d'Ω toutes les opérations restent internes, de sorte que nous avons toute liberté pour définir des caractères : en ISO 10646/UNICODE, la zone privée est composée des caractères `0xe000` à `0xfffd` (du groupe 0, qui est la partie 16 bits de ISO 10646/UNICODE), soit un total de 8190 positions.

Ω traitera l'entrée comme dans la section précédente, mais les lettres avec macron et brève occuperont des positions internes dans la zone privée. Le reste du traitement sera exactement le même. Comme pour la translitération de l'entrée, on peut prendre ^ et _ pour noter macron et brève (après avoir changé leur *catcode* pour éviter toute inter-

[7] Ces signes diacritiques supplémentaires sont aussi utilisés avec un autre objectif : en prose, placés après l'une des lettres alpha, iota ou upsilon, ils indiquent s'il faut prononcer cette dernière longue ou courte (cette fois nous parlons de *lettres* et pas de *syllabes*).

[8] Néanmoins, ils ont introduit les lettres alpha, iota et upsilon avec macron et brève, en minuscules et majuscules, probablement pour les raisons exposées dans la note précédente. Pourtant, la combinaison de diacritiques doit être utilisée pour coder des lettres avec macron/brève *et* des diacritiques supplémentaires.

férence avec les opérateurs mathématiques), ou n'importe quelle combinaison de codes 7 ou 8 bits.

2.5. Un rêve qui *peut* devenir réalité

Comme le premier auteur l'a declaré déjà à Cork en 1990, son rêve était — et demeure — de dessiner une fonte grecque inspirée du fameux « Grecs du Roi » de Claude Garamont, gravé en 1544-46 pour le roi François I. Ce caractère a été conçu d'après un manuscrit de Ἄγγελος Βεργήκιος, un Crétois, calligraphe et lecteur de grec à la cour française, au début du XVI siècle. Il comprend 1327 caractères différents, la plupart d'entre eux étant des ligatures de deux lettres ou plus (parfois des mots entiers). On peut lire dans [12] que « *cette police est la pièce la plus précieuse de la collection [de l'Imprimerie Nationale française]* », et ce n'est sûrement pas le moindre des honneurs ! Ω est la plateforme idéale pour composer avec cette police, puisque seul un ΩTP supplémentaire sera nécessaire pour introduire les ligatures dans le texte grec ordinaire de l'entrée.

2.6. L'arabe, ou « l'art de séparer les tâches ».

2.6.1. L'arabe ordinaire, rapide, net et élégant.

La composition en arabe est un bon compromis entre les techniques de composition occidentales (nombre fini de caractères, répétés à l'infini) et la calligraphie arabe (nombre infini de ligatures arbitrairement complexes). Nous pouvons subdiviser les ligatures arabes en deux catégories : (a) celles qui sont obligatoires : liaison de lettres (ه + م → هم) et la ligature spéciale lām-alīf (ل + ا → لا), et (b) celles qui sont optionnelles, utilisées pour des raisons esthétiques.

La deuxième catégorie de ligatures correspond à nos bons vieux « fi », « fl », etc. Elles dépendent du dessin de la police et du degré de qualité artistique du document. Nous avons fait une classification approfondie des ligatures esthétiques de la casse égyptienne [3], republiée dans [6]. Voici un exemple du processus de ligature du mot تحمل, en suivant les traditions typographiques égyptiennes :

— ت ح م ل (lettres disjointes) ;

— تحمل (ligatures obligatoires seulement, lettres reliées) ;

— تحمل (ligature esthétique entre les deux premières lettres) ;

— تحمل (ligatures esthétiques entre les trois premières lettres) ;

Pour produire toutes les ligatures possibles (plus de 1500) de deux, trois ou quatre lettres, il a fallu trois tableaux de 256 caractères. Chaque ligature est construite par superposition de petits éléments. Une fois que TEX sait quels caractères prendre, et dans

quelle fonte, il lui suffit alors de les superposer (aucun déplacement n'est nécessaire). Le problème est de reconnaître l'existence d'une ligature et de découvrir quels sont les caractères nécessaires. Ce processus dépend fortement des fontes. Une fonte différente — par exemple en style Kuffic ou Nastaliq — peut avoir un ensemble de ligatures complètement différent, ou aucune ligature (comme la police de base, dans laquelle sont écrits les deux mots تيخ العربى, et qui est largement utilisée en composition électronique grâce à sa lisibilité) ; néanmoins, les ligatures obligatoire restent exactement les mêmes, quelle que soit la fonte utilisée.

Jusqu'à maintenant, il existe trois solutions au problème des ligatures arabes obligatoires :

— La première, par K. Lagally [10], consiste à utiliser des macros TEX pour détecter et appliquer les ligatures obligatoires (dans notre terminologie : « faire l'analyse contextuelle »). Ce processus est malcommode et long. Il dépend beaucoup du codage de la fonte et les macros utilisées peuvent interférer avec d'autres macros TEX. Tout compte fait, ce n'est pas la façon naturelle de traiter un phénomène qui est une caractéristique fondamentale de l'écriture arabe.

— La deuxième, par le premier auteur [5], consiste à utiliser les ligatures propres à TEX (couplées avec TEX–XET, la version bidirectionnelle de TEX) ; sur le fond, ce processus est plus naturel, puisque l'analyse contextuelle est faite « dans les coulisses », au plus bas niveau, à savoir celui des fontes lui-même. Il ne dépend pas du codage de la police, puisque chaque fonte peut utiliser son propre jeu de ligatures. L'inconvénient réside dans le nombre de ligatures nécessaire pour réaliser la tâche : environ 7000 ! La situation devient dramatique quand on veut utiliser une douzaine de polices arabes sur la même page : TEX chargera 7000 ligatures probablement strictement identiques pour chaque police. Vous aurez besoin de bien plus qu'un BigTEX pour cela.

— La troisième, également par le premier auteur [5], utilise un préprocesseur. L'intérêt en est que l'analyse contextuelle est réalisée par un utilitaire dédié à cette tâche, avec plusieurs fonctionalités supplémentaires (par exemple l'ajout de jonctions de longueur variable, aussi appelées « keshideh ») ; c'est une méthode rapide qui utilise très peu de mémoire. Cette méthode présente malheureusement les inconvénients classiques des préprocesseurs traitant un document avant TEX : un fichier peut en inclure un autre (`\input`), depuis un endroit quelconque de votre réseau, et vous ne pouvez pas savoir à l'avance quels fichiers seront lus, et donc devront être prétraités ; les directives du préprocesseur peuvent interférer avec les macros TEX ; il n'y a pas d'imbrication entre elles, ce qui peut facilement entraîner des erreurs par rapport aux opérations de groupage de TEX, etc.

Aucune de ces méthodes ne peut être appliquée à une production à grande échelle d'arabe dans la vie réelle : dans tous les cas, l'écriture arabe est traitée comme un « puzzle à résoudre » et, inévitablement, les performances de TEX en ressentent.

Nous utilisons des ΩTP pour fournir une solution naturelle au problème ; considérons à nouveau l'exemple du mot تجمل :

— D'abord, تحمل est lu par Ω, en translitération latine (`tHml`) ou en ISO 8859-6, ou en ASMO, ou en codage Macintosh arabe, ou en un quelconque codage d'entrée arabe acceptable.

— Le premier ΩTP convertit cette entrée en codes ISO 10646/UNICODE pour lettres arabes : `0x062a` (ت, ARABIC LETTER TEH), `0x062d` (ح, ARABIC LETTER HAH), `0x0645` (م, ARABIC LETTER MEEM), `0x0644` (ل, ARABIC LETTER LAM) ;

— ISO 10646/UNICODE étant une façon *logique* de coder les lettres arabes, et pas une façon graphique, il n'y a pas d'information sur leur forme contextuelle (isolée, initiale, médiale, finale). Le deuxième ΩTP envoit ces codes à la zone privée, où nous avons (de façon interne) réservé des positions pour la combinaison de caractères arabes et de formes contextuelles. Une fois que c'est fait, Ω connaît la forme de chaque caractère.

— Le troisième ΩTP traduit simplement ces codes dans un codage police TEX standard 16 bits (c'est une opération mineure : la zone privée étant située à la fin du tableau 16 bits, nous déplaçons tout le bloc au début du tableau).

— Si la fonte n'a pas de ligatures esthétiques, c'est fini : Ω va envoyer les résultats du dernier ΩTP au fichier `dvi`, et produire تحمل. D'un autre côté, s'il y a encore des ligatures esthétiques — comme dans تحمل — alors elles seront insérées dans la fonte, en tant que ligatures intelligentes (*"smart ligatures"*). Puisque les tables des fontes peuvent contenir jusqu'à 65536 caractères, on a toute la place nécessaire pour mettre les petites parties caractères à combiner.[9]

  Ce que nous avons réalisé consiste à ce que le processus fondamental d'analyse contextuelle soit traité par une machinerie en arrière-plan (exactement comme TEX fait la césure et divise les paragraphes en lignes), et que les raffinements esthétiques éventuels soient exclusivement gérés par les fontes (par analogie avec les fontes romaines qui ont plus de ligatures que celles pour machine à écrire, etc).

2.6.2. L'arabe voyellisé (les choses deviennent plus difficiles)

Dans l'arabe contemporain ordinaire, on écrit seulement les consonnes et les voyelles longues ; le lecteur doit deviner les voyelles courtes, en utilisant le contexte (les mêmes consonnes avec des voyelles courtes différentes peuvent être comprises comme verbe, comme nom ou adjectif etc.). Quand il est essentiel de spécifier des voyelles courtes, on ajoute de petits signes diacritiques au-dessus ou au-dessous des lettres.

À côté des voyelles courtes, il y a aussi des diacritiques pour les consonnes doubles, pour indiquer l'absence de voyelle, et pour le coup de glotte (comme dans « Oh-oh ») : en comptant toutes les combinaisons, nous arrivons à 14 signes. Ces diacritiques peuvent

[9] Si ces ligatures esthétiques sont utilisées dans plusieurs polices, on pourrait rencontrer le même problème de surcharge de la mémoire fonte de Ω ; dans ce cas, on peut toujours écrire un quatrième ΩTP, qui ferait systématiquement l'analyse « esthétique » indépendamment des codes d'analyse contextuelle.

rendre la vie difficile à TEX, car ils doivent être codés entre les consonnes, et interviennent ainsi dans l'algorithme d'analyse contextuelle : supposons par exemple que TEX doive composer la lettre *x*, dernière lettre d'un mot, suivie d'un point. Après la lecture du point, TEX sait que la lettre doit être sous forme finale (l'une des 7000 ligatures doit être <*x* sous forme médiale> + < . > → <*x* sous forme finale>< . >). Supposons maintenant que la lettre soit immédiatement suivie d'une voyelle courte, qui dans notre cas est forcément placée entre la lettre *x* et le point. Les ligatures intelligentes de TEX ne peuvent revenir deux positions en arrière ; quand TEX découvre le point après la voyelle courte, il est trop tard pour convertir le *x* médial en un *x* final.

Heureusement, les ΩTP sont suffisamment ingénieux pour déterminer la forme des lettres, quelles que soient les diacritiques qui les entourent (ce qui est exactement l'attitude d'un typographe humain, qui compose d'abord les lettres, puis ajoute les diacritiques correspondantes).

Néanmoins, les ΩTP ne sont pas parfaits, et il y a des problèmes impossibles à résoudre même avec les ΩTP les plus efficaces : le positionnement des diacritiques par exemple. Nous savons tous que TEX (et par conséquent Ω, qui n'est rien de plus qu'une humble extension de TEX) place tous les éléments sur la page en utilisant des boîtes. Malheureusement, le positionnement des diacritiques nécessite plus d'informations que les seules hauteur, largeur, profondeur et correction d'italique d'un caractère ; dans certains cas, un véritable examen de la forme du caractère lui-même et des caractères adjacents est nécessaire (pensez aux ligatures construites verticalement à partir de quatre lettres, chacune ayant sa propre diacritique).

Ce problème peut facilement être résolu pour une fonte sans ligature (esthétique) : en comptant toutes les lettres possibles (sans oublier le farsi, l'ourdou, le pashto, le sindhi, le kirghiz, l'ouigoure et d'autres langues utilisant des variantes des lettres arabes), dans toutes les formes possibles, on peut arriver à un nombre ne dépassant pas 1000 glyphes. En combinant ces glyphes avec les 14 signes diacritiques, on atteint moins de 14 000 positions, un chiffre bien en-dessous de la limite de 65 536 caractères. Puisque la zone privée de ISO 10646/UNICODE n'est pas assez grande pour gérer autant de caractères, nous utiliserons un ΩTP supplémentaire pour envoyer des combinaisons de <consonnes ou voyelles longues analysées contextuellement> et <diacritique> vers des codes du codage fonte en sortie. L'avantage de cette méthode est de permettre de placer individuellement chaque diacritique (en comptant une minute pour trouver la position idéale d'un signe diacritique, la fonte peut être terminée en quatre semaines de travail régulier), mais on peut utiliser des méthodes QDTEXVPL pour placer automatiquement les diacritiques, puis faire les corrections nécessaires.

Malheureusement le nombre de positions nécessaires dans les fontes croît de façon astronomique quand nous considérons des ligatures de 2 ou 3 lettres. Un des futurs défis du projet Ω sera d'analyser les caractères manuscrits arabes et de trouver les paramètres nécessaires pour déterminer le positionnement des diacritiques, exactement comme D.E. Knuth l'a fait pour la composition mathématique. Il faut noter que malgré l'immense complexité de cette tâche, nous restons dans le strict domaine de la *typo-*

graphie arabe, qui n'est après tout rien de plus qu'une *version réduite et simplifiée de la calligraphie arabe*.

2.7. Langues à plusieurs systèmes d'écriture (« savez-vous *lire* le vietnamien ? »)

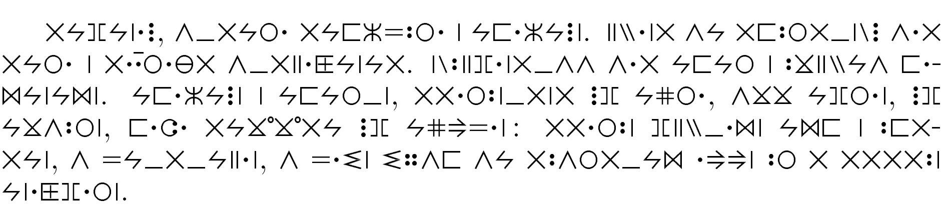
Les Occidentaux comme les Arabes ont eu l'habitude — pas vraiment démocratique — d'imposer leur alphabet aux nations dont ils faisaient la conquête (de façon militaire, religieuse, culturelle ou technologique). Aussi arrive-t-il que nous puissions lire le viêtnamien (mais sans le comprendre) tout comme n'importe quel Arabe peut lire le malais et le sindhi, et ne pas en comprendre un mot (sauf peut-être quelques mots arabes qui ont accompagné l'alphabet dans ses voyages).

Dans certains cas, plusieurs écritures sont restées en usage pour la même langue et des efforts sont faits pour éclaircir et standardiser les équivalences entre les lettres de ces différentes écritures, en vue de fournir un algorithme efficace de translitération.

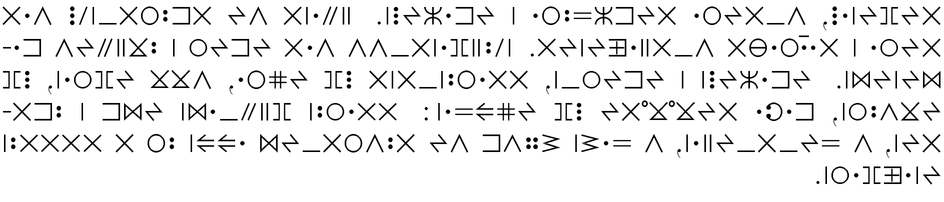
Le premier auteur a travaillé sur deux cas qui concernent l'écriture arabe : le berbère et le comorien.

2.8. Le berbère : une langue à trois systèmes d'écriture et deux directions d'écriture

La langue berbère (les Berbères l'appellent le « tamazight ») peut être écrite en arabe, en latin ou en écriture native (« tifinagh »). Le premier auteur a développé, sous la direction de Salem Chaker, directeur du département d'études berbères de l'Institut National de Langues Orientales à Paris, une police tifinagh en METAFONT. Voici un petit exemple de texte berbère, écrit en tifinagh, de gauche à droite.



Le même texte en tifinagh, mais de droite à gauche :



et en écriture arabe :

تيفيناغ، دتير، تيمزوّر، ن يِمازيغن. لّانت دي تمورتنّغ د،ت تير، ن تاعر،بت دتلاطينيت. نّولفانتدّ د،ت يِمير ن وقلّيد ماسينيسن. يِمازيغن ن يِميرن، تّارون‌تنت غف يِزر،، دقّ يِفر،ن، غف يِقدورن، ماشا تيقتّتي غف يِزكو،ن : تّارون فلّ،سن يِسم ن ومتّين، د ويتيلان، د و،ين يخدم دي تودرتيس ،كّن ور ت تّتّون يِناطفارن.

Le codage de ces fontes est conçu de sorte qu'on puisse traiter la même entrée TEX en translitération latine, en tifinagh de gauche à droite, de droite à gauche ou en arabe. Il suffit de changer une macro au commencement du traitement. Réaliser cette fonctionnalité a été plus ou moins simple pour le latin et le tifinagh, mais pas autant pour l'arabe. Malheureusement, cette police présente tous les problèmes des fontes arabes ordinaires : il faut plus de 7 500 ligatures pour faire l'analyse contextuelle, et elle est surchargée : il n'y a plus la place d'ajouter un seul caractère, un inconvénient pour une langue qui est encore en train d'être standardisée.

Une autre source de difficultés réside dans le fait que les équivalences entre l'écriture latine, tifinagh et arabe ne sont pas immédiates. Certaines voyelles courtes sont écrites dans le texte latin, mais pas dans celui en arabe ou en tifinagh. De plus, les consonnes doubles sont écrites explicitement en latin et tifinagh, mais comme une consonne simple en arabe. Et peut-être le problème le plus difficile à résoudre est de faire en sorte que chaque écrivain berbère se sente « chez lui », quelle que soit l'écriture qu'il utilise : il ne doit pas avoir l'impression qu'une écriture est privilégiée par rapport aux deux autres !

Finalement, le dernier problème (qui n'est pas le moindre quand il se pose dans le monde de la production) est que nous avons besoin d'une police arabe spéciale pour le berbère, à cause de la différence de translitération en entrée : par exemple, alors qu'avec la translitération arabe ordinaire nous utilisons « v » pour ڤ et « sh » pour ش, en berbère nous sommes obligés d'utiliser « g » pour le premier et « c » pour le dernier. Il y a deux lettres supplémentaires utilisées pour le berbère en écriture arabe : ڜ et ڹ ; ces lettres sont aussi utilisées en sindhi et pachtou, de sorte que les glyphesexistent déjà dans le système TEX standard arabe ; mais en berbère, ils doivent être transformés par translitération en « j » et « z », à cause des équivalences avec l'alphabet latin. Ceci nous force à utiliser un schéma de translitération différent de celui prévu pour l'arabe ordinaire, et donc — en raison de l'incapacité de TEX à séparer clairement les codages entrée et sortie — d'utiliser une fonte de sortie TEX codée différemment. Supposez que vous soyez en train de composer un livre à la fois en berbère et en arabe ; vous aurez besoin de deux polices graphiquement identiques pour chaque style, corps, graisse et famille, chacune d'entre elle ayant plus de 7 000 ligatures. Et nous ne parlons que de fontes sans ligatures esthétiques !

Ω résoud ce problème en utilisant les mêmes fontes en sortie pour l'arabe ordinaire et le berbère. Nous devons simplement remplacer le premier ΩTP de la chaîne

de traduction : celui qui convertit l'entrée brute en codes ISO 10646/UNICODE. Les linguistes berbères sont libres d'inventer/introduire de nouveaux caractères ou signes diacritiques ; pour peu qu'ils soient inclus dans le tableau ISO 10646/UNICODE, il nous suffit de changer légèrement le premier ΩTP (et si ces signes ne sont pas encore dans ISO 10646/UNICODE, nous utiliserons la zone privée).

2.9. Le comorien : écriture latine africaine ou arabe

La situation est similaire dans les petites îles des Comores, entre Madagascar et le continent africain. Les alphabets latin (avec quelques adjonctions venant de langues africaines) et arabe sont utilisés. À cause des nombreux sons qu'il faut différencier, on utilise des diacritiques accompagnant des lettres arabes. Ces diacritiques ressemblent aux diacritiques arabes (pour des raisons pratiques) mais elles ne sont pas utilisées de la même façon ; en fait, elles font partie des lettres, exactement comme les points font partie des lettres de l'arabe ordinaire.

Une fois encore, la situation peut facilement être gérée par un ΩTP. Alors que les propositions d'insertion dans ISO 10646/UNICODE ne sont pas encores claires (la proposition faite par Ahmed-Chamanga, de l'Institut des Langues Orientales à Paris, circule maintenant entre ministères et institutions éducatives et religieuses), les Comoriens peuvent déjà utiliser Ω pour faire la composition et faire progresser le schéma de translitération à la volée.

2.10. Le cambodgien

Comme noté dans [4], l'écriture cambodgienne utilise des groupes consonantiques, des consonnes souscrites, des voyelles et des signes diacritiques. À l'intérieur d'un groupe, TEX doit déplacer les différents composants pour les positionner correctement. Il en résulte que TEX *doit* utiliser des commandes `\kern` entre chaque composant d'un groupe. Aussi n'y a-t-il plus de crénage : supposons que les caractères ្ក et ្ម doivent être crénés ; et supposons que la consonne ក est (logiquement) suivie de la consonne souscrite ្ន, qui est (graphiquement) placée sous cette lettre : ក្ន ; ្ម ne suit plus immédiatement ក, et ainsi il n'y aura aucun crénage entre ces lettres ; néanmoins, graphiquement elles sont toujours adjacentes, et doivent donc être éventuellement crénées.

Ω utilise une méthode marteau-pilon pour résoudre ce problème : nous définissons une « grande » fonte cambodgienne (virtuelle), contenant *tous les groupes actuellement connus*. Comme nous l'avons déjà mentionné dans [4], à peu près 4 000 codes devraient suffire. On peut évidemment encore utiliser les méthodes traditionnelles de TEX pour former des groupes consonantiques exceptionnels, non contenus dans cette police.

Comme en arabe, nous venons à bout de la complexité du cambodgien en séparant les tâches. Un premier ΩTP enverra la méthode d'entrée que l'utilisateur a choisie vers les codes cambodgiens d'ISO 10646/UNICODE (en réalité) il n'existe encore aucun code cambodgien ISO 10646/UNICODE, mais le premier auteur a soumis une proposition de

codage cambodgien aux comités ad-hoc de l'ISO, et espère qu'il y aura bientôt des avancées dans cette direction — pour l'instant nous utiliserons une fois de plus la zone privée). Un deuxième ΩTP analysera contextuellement ces codes et les groupera avant de les envoyer aux codes de groupes consonantiques appropriés. La séparation des tâches est essentielle pour autoriser des méthodes d'entrée multiples, sans redéfinir à chaque fois l'analyse contextuelle — ce qui après tout est une caractéristique de base du système d'écriture cambodgienne. Ω enverra le résultat du deuxième ΩTP au fichier `dvi`, en utilisant l'information de crénage contenue dans la police (virtuelle). Finalement, `xdvicopy` dé-virtualisera le fichier `dvi` et créera un nouveau fichier utilisant exclusivement des caractères de la police cambodgienne originale 8 bits, décrite dans [4].

2.11. ISO 10646/UNICODE et après

C'est assurément un tâche non triviale d'adapter des caractères d'alphabets différents pour obtenir un résultat optiquement homogène. Souvent l'esthétique inhérente aux différents systèmes d'écriture ne permet pas suffisamment de manipulation pour les rendre « ressemblantes » ; ce n'est pas sans importance de savoir s'il faut l'essayer en premier : supposons que vous choisissiez l'hébreu et l'arménien et que vous modifiez la forme des lettres jusqu'à ce qu'elles se ressemblent suffisamment, à nos yeux d'occidentaux. Il n'est pas évident que l'arménien ressemble encore à de l'arménien, ou que l'hébreu ressemble à de l'hébreu ; en outre il ne faut pas négliger la nécessité pour le lecteur de sauter d'une écriture à l'autre (avec tous les autres changements que le passage de l'une à l'autre implique : langue, culture, état d'esprit, idiosyncrasie, formation) : plus l'écriture diffère, plus la transition peut être rendue facile.

La seule chose sûre que nous puissions faire avec des caractères d'origine différente est de compenser l'épaisseur du trait de sorte que sur la page, la densité globale de gris soit homogène (pas de « trous » à l'intérieur du texte, chaque fois que nous changeons d'écriture).

Ces remarques concernent d'abord les écritures qui ont une esthétique très différente. Il y a un cas pourtant où l'on peut appliquer tous les moyens d'uniformisation, et où les lettres peuvent immédiatement être reconnues comme faisant partie de la même famille de fontes : le groupe LGCAI (LGCAI signifie « Latin, Grec, Cyrillic ocidental/oriental, Africain, Vietnamien et IPA »). Très peu de familles couvrent le groupe entier : Computer Modern est l'une d'entre elles (pas vraiment la plus belle), Unicode Lucida en est une autre (belle police latine mais plutôt un échec dans le cas du le grec minuscules) ; il y a des polices Times pour toutes les langues du groupe, mais sans garantie qu'elles appartiennent au même style Times, comme pour Helvetica et Courier. D'autres adaptations ont également été tentées et on peut s'attendre à ce que le succès (?) de Windows NT conduise d'autres fondeurs à « étendre » leur fontes à l'ensemble du groupe[10]

[10] Comme le faisait remarquer un africain autochtone au premier auteur, cela signifie aussi que les Africains vont se trouver dans la situation désagréable et paradoxale d'avoir (a) des fontes pour leurs langues, (b) des ordinateurs, puisque les universités occidentales envoient tous leurs vieux équipements au tiers-monde, mais (c) pas d'électricité pour les faire fonctionner et utiliser les fontes ...

Heureusement, les utilisateurs de TEX/Ω peuvent dès aujourd'hui composer dans l'ensemble des langues du groupe LGCAI, en Computer Modern[11] (en ajoutant éventuellement quelques caractères et en en corrigeant certains autres). Les tableaux de fontes 16 bits de Ω permettent d'avoir :

1. des motifs de césure utilisant des caractères arbitraires du groupe ;
2. la possibilité d'éviter des changement trop fréquents de fonte, par exemple en passant du turc au gallois, au viêtnamien, à l'ukrainien, au haoussa ;
3. un crénage potentiel entre tous les caractères.

Mais Ω va encore au-delà : on peut inclure différents styles dans la même fonte virtuelle ; en examinant la table ISO 10646/UNICODE, on voit que les caractères LGCAI, avec en plus tous les dingbats et signes de ponctuation dépendant du style, tiennent dans 6 rangées (1 536 caractères). Cela signifie que — au moins théoriquement — une fonte virtuelle Ω peut contenir jusqu'à 42 (!) variations de style[12] du groupe LGCAI entier, par exemple italique, gras, petites capitales, sans sérif, machine à écrire, plus toutes les combinaisons possibles [un total de 24 = (romain ou italic) $\times$ (normal ou gras) $\times$ (normal ou petites capitales[13]) $\times$ (sérif ou sans sérif ou machine à écrire)]. La définition de paires de crénage entre tous ces styles différents évitera l'usage et l'abus de $\backslash/$ (correction d'italique) et améliorera l'apparence de texte utilisant plusieurs styles.

Ce sera une véritable expérience de réaliser une telle fonte, puisque beaucoup de caractères africains et IPA n'ont pas encore de style pour les majuscules, les petites capitales ou l'italique ; on consultera le papier de Jörg Knappen sur les fontes africaines [9] pour la description de quelques exemples posant des problèmes et des solutions qu'il propose.

2.12. Autres applications : *Ars Longa, Vita Brevis*

Dans ce papier, nous avons choisi certaines applications d'Ω, en dehors de tous critères personnels ou subjectifs. Presque chaque langue ou écriture peut profiter d'une façon ou d'une autre des possibilités offertes par les processus internes de traduction et les tables 16 bits. Par exemple, l'un des auteurs (cf. [7]) a présenté en 1994 au congrès du TUG le pré-processeur *indica* pour les langues indiennes (langues du sous-continent indien, sauf l'ourdou). *Indica* sera réécrit comme un ensemble d'ΩTP ; de cette façon, nous éliminerons tous les problèmes de pré-traitement de code TEX.

Tout compte fait, le développement de la composition 16 bits va être un défi fascinant pour la prochaine décennie, et Ω peut y jouer un rôle important, à cause du soutien du

[11] *Sans aucun doute, tôt ou tard une institution ou une société prendra l'initiative digne d'éloges de parrainer le développement d'autres familles METAFONT* ; les auteurs voudraient promouvoir cette idée.

[12] Les auteurs voudraient souligner le fait qu'il n'est aucunement nécessaire de produire tous les styles à partir du même code METAFONT, comme cela a été fait pour Computer Modern. En fait la police dont nous parlons peut très bien être un mélange de Times, Helvetica, Courier ; l'avantage de les avoir dans la même structure est que nous pouvons définir des paires de crénage entre caractères de styles différents.

[13] Le chiffre est en fait inférieur à 18, puisque seules les petites capitales en minuscules sont nécessaires...

monde universitaire et de la recherche, de son aspect ouvert, de sa portabilité et de son caractère non commercial.

Bibliographie

[1] B. Beeton. Mathematical symbols and Cyrillic fonts ready for distribution (revised). *TUGboat*, 6(3):124–128, 1985.

[2] G. Betts and A. Henry. *Teach yourself Ancient Greek*. Hodder and Stoughton, Kent, 1989.

[3] Y. Haralambous. Typesetting the holy Qur'ān with TEX. In *Proceedings of the 2nd International Conference on Multilingual Computing—Arabic and Latin script (Durham)*, 1992.

[4] Y. Haralambous. The Khmer script tamed by the lion (of TEX). In *Proceedings of the 14th TEX Users Groups Annual Meeting (Aston, Birmingham)*, 1993.

[5] Y. Haralambous. Nouveaux systèmes arabes TEX du domaine public. In *Comptes-Rendus de la Conférence « TEX et l'écriture arabe » (Paris)*, 1993.

[6] Y. Haralambous. Typesetting the holy Qur'ān with TEX. In *Comptes-Rendus de la Conférence « TEX et l'écriture arabe » (Paris)*, 1993.

[7] Y. Haralambous. *Indica*, a preprocessor for indic languages—Sinhalese TEX. In *Proceedings of the 15th TEX Users Groups Annual Meeting (Santa Barbara)*, 1994.

[8] Y. Haralambous and J. Plaice. First applications of Ω: Greek, Arabic, Khmer, Poetica, ISO 10646/UNICODE, etc. In *Proceedings of the 15th TEX Users Groups Annual Meeting (Santa Barbara)*, 1994.

[9] J. Knappen. Fonts for africa. *TUGboat*, 14(2):104–106, 1993.

[10] K. Lagally. ArabTEX. In *Proceedings of the 7th European TEX Conference (Prague)*, 1992.

[11] S. Levy. Using greek fonts with TEX. *TUGboat*, 9(1):20–24, 1988.

[12] Imprimerie Nationale. *Les caractères de l'Imprimerie Nationale*. Imprimerie Nationale, Éditions, Paris, France, 1990.

[13] J. Plaice. Progress in the Ω project. In *Proceedings of the 15th TEX Users Groups Annual Meeting (Santa Barbara)*, 1994.

[14] D. Vulis. Notes on Russian TEX. *TUGboat*, 10(3):332–336, 1989.