SAGACE v4.2.0

2012/07/10

R.BLIN (CNRS-CRLAO)

Table des matières

A. Présentation

SAGACE est un logiciel de recherche de chaînes de caractères dans des textes bruts.

Il permet

- d'extraire des segments de textes (par exemple des phrases) contenant un morphe ou une suite donnée de morphes (fonction de concordancier).

- de lister des collocations et éventuellement de les compter.

- de gérer un lexique.

Nous détaillons le fonctionnement du logicel dans les lignes suivantes et le comparons aux logiciels existants.

Il est gratuit et libre sous licence CECILL, librement distribué à l'adresse :

`http : //crlao.ehess.fr/japonais-coreen/corpus/sagace/sagace_telechargement.html`

A.1. Types de tâches effectuées

1) Concordancier : extraction de segments de textes comprenant une chaîne continue ou discontinue déterminée

A partir d'un corpus tagué ou d'un texte brut, le programme extrait tous les segments de textes contenant une chaîne (continue ou discontinue) de morphes. Il est possible de chercher un morphe particulier ou tout morphe appartenant à une catégorie donnée par l'utilisateur. Le choix du corpus, la nature du segment (phrase, paragraphe, ou autres) à extraire, la définition des catégories syntaxiques dans le lexique sont faciles à définir et peuvent être modifiées aisément par l'utilisateur. Les résultats sont enregistrés dans un fichier de résultats dont la présentation est paramétrable.

L'utilisateur peut aussi demander d'extraire le segment qui précède.

Par exemple, la catégorie des verbes japonais ayant été préalablement donnée par l'utilisateur, on cherche toutes les phrases comprenant un verbe composé dont le premier verbe est à la forme *renyô*.

2) Collocations

Le programme relève toutes les collocations répondant au caractéristiques spécifiées par l'utilisateur. Il peut en plus faire un comptage.

En reprenant l'exemple précédent, le résultat sera une liste des verbes composés dont le premier verbe est à la forme ren'yô.

3) Manipulation de lexique

La recherche exploite si nécessaire un lexique dans lequel sont définies les catégories. Le logiciel permet une manipulation rudimentaire en ligne de ce lexique.

A.2. A quoi peut servir SAGACE ?

SAGACE peut être exploité dans plusieurs domaines.

1) Linguistique

SAGACE permet au linguiste de faire des recherches de structures beaucoup plus complexes que ce que permettent les moteurs de recherches généralistes disponibles sur l'internet. Les résultats sont par ailleurs incomparablement plus fiables.

2) Enseignement

Le logiciel offre la possibilité à l'enseignant, mais aussi à l'étudiant, de composer des listes de vocabulaires pertinents, d'en étudier et illustrer les usages en listant des exemples. Blin (2011, à paraître) montre comment composer un lexique d'apprentissage, ou évaluer un lexique existant.

3) Lexicographie

Suffisamment souple, le logiciel peut aussi être utilisé pour étudier les propriétés d'un lexique, comme par exemple étudier la corrélation entre des phénomènes phonologiques dans un lexique (Blin 2010). Par exemple, sous réserve de donner au lexique la forme adéquat, il est possible d'étudier les propriétés phonologiques de la strate lexicale des kango.

4) Intégration de SAGACE dans des logiciels

Les résultats obtenus par SAGACE étant enregistrés dans un format ouvert (du texte) et SAGACE étant commandé en ligne de commande et par des fichiers de configuration ouvert (du texte), il est tout à fait possible de l'intégrer dans un processus complexe d'analyse de texte, en récupérant ses résultats (Blin R., JEC, 2011 à paraître).

A.3. Historique

Le projet a démarré dans les années 90 et a été officiellement déclaré à l'APP en 2004 (version 1.0.0).

SAGACE a été conçu isolément, sans tenir compte du contexte qu'il était difficile de suivre à l'époque, du fait de la stricte compartimentation des systèmes informatiques (Windows, Mac, UNIX, et même (... et surtout...) à l'intérieur même de la famille Windows, il existait des machines Windows Japon et le reste), des matériels (par exemple les disquettes Japon étaient formatées en 1.2 Mo tandis qu'elles l'étaient en 1.4 en France), de la compartimentation des systèmes d'encodage des caractères, et en fin du fait de la difficulté en l'absence d'un Internet significativement développé.

L'objectif était initialement de se doter pour l'analyse linguistique du japonais d'un logiciel de manipulation de textes japonais capable d'extraire des exemples, et qui soit simple et très paramétrable.

Il est apparu dès le départ que le logiciel devait pouvoir travailler sur des textes non balisés, de sorte à être exploitable immédiatement sur un texte quelconque, sans traitement préalable de ce texte. L'autre intérêt de travailler sur du texte non balisé est de ne pas dépendre d'une théorie morpho-syntaxique donnée. En effet, pour exploiter un texte balisé, il faut s'en tenir au catégories morpho-sémantique utiliées pour le balisage, ce qui ne convient pas nécessairement lorsqu'au cours d'une recherche, on veut au contraire expérimenter des catégories originales.

Le logiciel devait aussi être indépendant du codage, problème sensible dans les années 90 et qui est en passe de disparaître avec la généralisation de l'unicode.

Un premier principe fondamentale de SAGACE est de chercher des motifs, au même titre par exemple que des logiciels de recherche d'expressions régulières (nous reviendrons sur ce fonctionnement en section B.3).

Le second principe est d'être complètement intégré : il "segmente" le texte, extrait ou compte, et affiche.

Le moteur n'a guère évolué depuis le début de sa conception. L'évolution a essentiellement porté sur l'ergonomie soit de la description des motifs, soit de l'affichage des résultats.

Le projet a connu une évolution très sensible en janvier 2008 avec l'introduction d'une représentation syntaxique à base de systèmes de traits au lieu d'une notation jusque là très lourde et peu maniable. La description des catégories par des système de traits a permi d'introduire un langage simple d'interrogation, qui donne une grande liberté de manoeuvre au requêteur et permet de créer des catégories "à la volée" à partir des données disponibles dans le lexique, mais sans avoir à intervenir sur ce lexique. C'est la version 2.0.0 de SAGACE.

Malgré tout, le langage utilisé pour décrire ces systèmes de traits était un langage *ad hoc* frustrant et limité. Début 2009, la "logique" de cette notation par traits est poussée à son terme. Il en sort une notation à l'aide de formules interprétables en logique des propositions. C'était la version 3 de SAGACE.

La version 3.2.X voit l'ajout d'un gadget : la possibilité de tenir compte de la hiérarchie des textes (niveau de titrage, portée des titres).

Du point de vue de l'affichage, le progrès notoire a été la possibilité d'afficher le concordancier au format KWIC.

Le logiciel conçu pour répondre à ces besoins pour le japonais s'est révélé suffisamment souple pour travailler sur n'importe quelle langue, sans modification du moteur, pourvu que la langue étudiée n'ait pas de flexions trop complexes.

L'usage de SAGACE a été étendu au coréen dès 2000 et au chinois (simplifié) en 2006. En fait, pour cela, le moteur de SAGACE n'a connu aucune modification en soi. Ce sont les lexiques qui supportent toutes les particularités des langues.

L'étude du coréen pose toutefois problème et en 2009, la construction du lexique est complètement suspendue. La faute revient au mode d'encodage des caractères coréens, qui sont graphiquement compositionnels mais que l'encodage ne permet pas de décomposer selon des règles manipulables par SAGACE, que ce soit l'encodage EUC-KR le plus répandu, ou UTF-8. Le traitement du coréen par SAGACE reste donc à améliorer, peut-être en passant par une transcription en caractères latins, qui serait « invisible » pour le manipulateur.

Le travail sur le chinois était fondamentalement dépendant de la constitution d'un lexique. Un lexique LEXS-CHS a été constitué en 2008 et amélioré. Toutefois, l'expérience n'est pas poursuivie car il n'y a plus de concepteur pour le dictionnaire.

La présente version 4, malgré un changement de numéro (de 3 à 4), ne présente pas de changement très significatif, mais plutôt consacre la désormais complète indépendance vis-à-vis de la langue traitée. En effet, dans les version 3, il restait une fonction de lemmatisation des verbes japonais qui est abandonnée dans la version 4. La lemmatisation est désormais traitée en exploitant les traits des représentations syntaxiques des entrées lexicales.

L'autre évolution est une plus grande rationalisation de l'affichage (voir section G.2.3.3).

B. Etat de l'art et positionnement de SAGACE

SAGACE ne se pose pas en concurrent des systèmes existants. Il est complémentaire de ce qui se fait. Nous proposons ici un comparatif en insistant sur les caractéristiques suivantes de SAGACE :

- autonomie et intégration
- légèreté du moteur
- recherche de motif
- faciilité de mise en oeuvre

B.1. Environnement intégré

Que ce soit pour les concordanciers ou l'extraction de collocations, les dispositifs sont conçus en trois modules : (1) segmentation du texte, (2) exploitation (comptage, repérage de l'environnement d'une chaîne) de cette segmentation, (3) affichage des résultats.

Il existe deux types d'approches, soit une approche intégrée où les trois étapes sont traitées par un même logiciel (indépendamment de sa structure interne), ou bien des suites ou "packages" de logiciels qui chacun traitent une étape.

SAGACE est un logiciel intégré qui effectue l'analyse (pas de "segmentation" à proprement parler), et l'affichage des données. Il est autonome et ne nécessite pas l'adjonction de logiciels extérieur.

Voici un comparatif avec d'autres dispositifs libres (logiciels libres et gratuits) exploités pour les mêmes fonctions (extraction de collocations et concordancier) sur le japonais ou susceptibles de l'être.

	sagace		INTEX (UNITEX/NOOJ)
Analyse	O	Chasen mecab KNP	o
Exploitation	o	o	o
Affichage	o	chakoshi chashi	o

L'intérêt de l'intégration du logiciel est essentiellement la facilité de gestion : on peut se le procurer en une seule fois, il n'y a pas de problème de compatibilités de versions etc.

B.2. Légèreté du logiciel

SAGACE est implémenté en C. Le moteur de recherche lui-même ne fait que 142 ko) et autonome puisqu'il ne nécessite pas le recourt à d'autres logiciels. Le code n'est pas optimisé et gagnerait à être retravaillé, pour gagner en vitesse d'exécution et de place.

B.3. Principe d'analyse

SAGACE ne procède pas à une analyse morpho-syntaxique complète d'un segment, mais s'en tient à détecter la présence d'un motif cherché, indépendamment de son environnement. Il s'apparente donc à des logiciels de recherche généralistes d'expressions régulières tel que grep, sed. Il est fondamentalement différent de logiciels qui effectuent des analyses complètes, tels que les outils statistiques comme Chasen, mecab, KNP, ou encore les logiciels de la famille INTEX (UNITEX ou NOOJ).

Un motif est décrit comme une suite continue ou discontinue de morphes ou ensembles de morphes (une catégories "morpho-syntaxiques"). Ainsi, on peut chercher le motif composé du morphe "私" suivi de "達 :

私 + 達

ou bien chercher un motif composé de la catégorie des pronoms et des suffixes du pluriel. Si ces catégories s'appellent respectivement "pronom" et "suffPluriel", le motif sera décrit :

pronom + suffPluriel

Les catégories sont définies dans un lexique. C'est à l'utilisateur de définir ses catégories (voir section F).

Par rapport aux fonctionnalités de logiciels comme sed ou grep, SAGACE ne dispose pas de beaucoup de souplesse dans sa description des formes du motif. Nous sommes en effet partis du principe que si l'on veut manipuler des motifs complexes, autant utiliser de véritables parseurs symboliques tels que UNITEX, NOOJ ou autres.

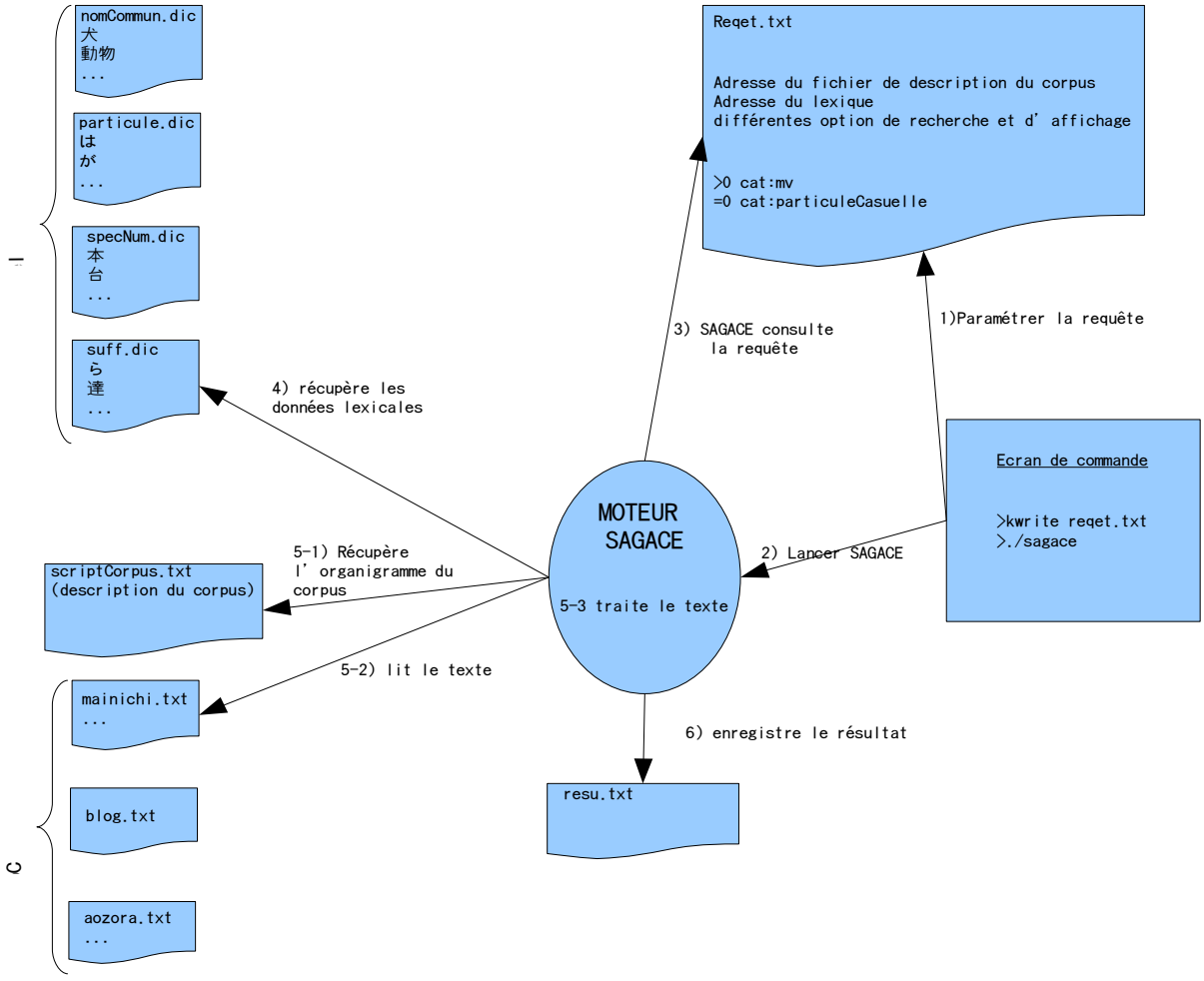
Par contre, un effort important a porté sur la description des composants du motif. Pour répondre au mieux aux besoins des recherches en linguistiques, il nous est apparu nécessaire de disposer d'un très volumineux lexique et de pouvoir le modifier à volonté. Comme il est hors de question de retravailler sans cesse un lexique qui peut comprendre des centaines de milliers d'entrées, SAGACE a été doté d'un langage de description des morphes du motif tel qu'il est bien sûr possible d'exploiter les catégories disponibles dans le lexique, mais aussi de créer des catégories " à la volée". Par exemple, mettons que le lexique dispose de deux catégories "nom" et "radical verbal" et qu'on veuille chercher un motif qui appartienne indifféremment à l'une ou l'autre de ces catégorie, le langage de description syntaxique des morphes du motifs permet de produire une union de catégorie : nom & radicalVerbal. Il est aussi possible de produire des catégories "intersections" ou encore des catégories "complémentaires". L'important est qu'il est possible d'effectuer ces opérations sur les catégories aussi bien en intervenant dans le lexique grammaire (ce sera une création "en dur") ou bien dans la description du motif. Dans ce cas, la modification n'est pas implémentée dans le lexique. L'intérêt de ce second type de modification est qu'il est possible d'expérimenter de nouvelles catégories sans mettre en danger la cohésion du lexique.

C. Fonctionnement général

La première utilisation de SAGACE demande un paramétrage. Il est possible de le modifier ultérieurement ou de le réutiliser en l'état.

Pour paramétrer correctement SAGACE, il est bon d'en connaître le fonctionnement dans ses grandes lignes.

SAGACE est constitué d'un moteur de recherche, d'un lexique, d'un corpus et d'un fichier de requête dans lequel sont indiqués les morphes-lemmes à chercher ainsi que quelques instructions complémentaires:



Lorsque SAGACE est lancé,

1) Il ouvre automatiquement et consulte un fichier contenant la requête (fichier décidé par l'utilisateur). Ce fichier sera désigné par le terme "*fichier de requête*" (chap.G, p.22). Dans ce fichier sont indiqués le type de recherche à effectuer (recherche d'exemples ou de collocations), l'emplacement du corpus et du lexique, ainsi que diverses options d'affichage des résultats. Y est aussi donnée la chaîne des morphes-lemmes à rechercher et leur distribution.

2) SAGACE charge en mémoire la partie du lexique nécessaire à la recherche (uniquement les lemmes-morphes à chercher).

3) Il lance ensuite la recherche sur le corpus et enregistre les résultats dans un fichier dont le nom est indiqué aussi dans le fichier de requête.

A l'occasion de la première utilisation, l'utilisateur devra
- Construire le corpus (section E).
- Construire le lexique (section F).

Ces deux sont à tout moment modifiables.

Puis pour chaque recherche (la première ou non), l'utilisateur doit
- Renseigner le fichier de requête (section G).

D. Installation

Le logiciel est distribué sous la forme d'un fichier compressé : `sagace-v4.X.X.tar.gz` . Les lignes ci-dessous correspondent à une installation sous linux.

Décompresser le fichier `tar.gz` dans le répertoire où vous désirez installer SAGACE.

```
tar ivf sagace-vX.Y.tar.gz
```

La décompression crée un répertoire sagace qui contiendra:

`sagace/`	
`sagace`	fichier exécutable
`reqet.txt`	fichier par défaut de paramétrage
`entete.txt`	entete par défaut du fichier html pour le format KWIC
`dico/`	répertoire par défaut destiné à recevoir les fichiers du dictionnaire
`corpus/`	répertoire par défaut destiné à recevoir les fichiers du corpus
`resu/`	répertoire par défaut destiné à recevoir les fichiers de résultats
`source/`	répertoire par défaut contenant les sources et un exécutable pour compiler
`doc/`	répertoire contenant le présent manuel et la licence

Si l'exécutable ne fonctionne pas, le logiciel peut être recompilé à partir des fichiers sources et de la commande contenu dans le script `source/sagace.prj` . Ce fichier est un script particulièrement basique. Il n'y a pas de `makefile`.

E. Construire un corpus

Le « corpus » désigne ici la collection de textes, balisés ou non, sur lesquels l'utilisateur travaille. Nous n'utilisons donc pas ici ce terme dans son sens plus restreint en TAL, où il désigne plutôt des textes balisés (pour une discussion rapide, voir Blin 2011 [[1]].

L'encodage du corpus est libre (shift-jis, unicode, etc...) mais doit être identique à celui du lexique et du fichier de requête.

E.1. Organisation du corpus

Matériellement parlant, le corpus est une collection de fichiers textes dont le nom, le nombre et l'organisation sont libres et laissés au choix de l'utilisateur. Les fichiers peuvent être enregistrés dans un unique répertoire ou dans différents répertoires et sous-répertoires. Ces répertoires ne doivent rien contenir d'autres que les fichiers du corpus car ils seront considérés comme des parties du corpus.

Exemple :

En japonais, l'utilisateur peut aisément se procurer le corpus de Aozorabunko[1] (textes littéraires libres de droits). Dans ce cas, il suffit de réunir tous les textes dans un sous-répertoire (ex : `aozorabunko`) du répertoire `corpus`. Il peut être intéressant de rassembler les textes par auteurs. Pour cela, créer un sous-répertoire portant le nom de l'auteur, dans lequel seront rassemblés les textes de l'auteur correspondant. Du coup le corpus a la forme :

```
/home/truc/corpus/aozorabunko/soseki/bochan
/home/truc/corpus/aozorabunko/soseki/waganeko
...
/home/truc/corpus/aozorabunko/dasai/truc
...
```

On peut tout à fait répartir les textes autrement.

E.2. Descripteur de corpus

A cet ensemble de fichiers est associé un fichier "*descripteur de corpus*" dans lequel sont indiqués les répertoires qui contiennent les fichiers du corpus.

Le chemin et le nom du descripteur de corpus est indiqué dans le fichier de requête (voir chap. G, p.22) comme suit :

```
ScriptCorpus:nom_et_chemin_du_descripteur_de_corpus
```

Par exemple :

```
ScriptCorpus:/home/truc/txt/txtJaponais/scriptCorpus.txt.
```

Le fichier descripteur de corpus peut être mis n'importe où, sauf sauf dans un des répertoires de corpus.

Dans le fichier descripteur de corpus, la description d'un corpus consiste à donner un nom au corpus, puis à lister les répertoires où figurent les fichiers (ne pas citer les fichiers). La syntaxe est la suivante :

```
groupe: nom_corpus1
répertoire
répertoire
...

groupe: nom_corpus2
répertoire
répertoire
...

groupe: nom_corpus3
```

1 www.aozora.gr.jp

```
répertoire
...
```

Attention, SAGACE ne consulte que les fichiers des répertoires indiqués mais pas les sous-répertoires. Il ne consultera les fichiers des sous-répertoires que si ceux-ci figurent dans la liste des répertoires à explorer.

Il faut impérativement mettre la barre oblique "/" a la fin du chemin.

Par exemple :

```
groupe: aozorabunko
/home/truc/corpus/txtJaponais/aozorabunko/
/home/truc/corpus/txtJaponais/aozorabunko/soseki/
/home/truc/corpus/txtJaponais/aozorabunko/dasai/

groupe: soseki
/home/truc/corpus/txtJaponais/aozorabunko/soseki/
```

Dans cet exemple, le premier corpus, appelé `aozorabunko`, réunit tous les textes de aozorabunko. Si SAGACE est lancé sur le corpus "aozorabunko", alors il consultera tous les fichiers qui se trouvent dans le répertoire `aozorabunko`, ainsi que tous ceux qui sont dans les sous-répertoires

`/.../aozorabunko/soseki/`

`/.../aozorabunko/dasai/`

Le second corpus, appelé *soseki*, réunit exclusivement les textes de Soseki, lesquels sont regroupés dans le répertoire `/.../aozorabunko/soseki/` . Lancé sur ce corpus, SAGACE n'étudiera que les textes de ce sous-répertoire et aucun autre.

Recourrir à plusieurs corpus permet de sélectionner rapidement les textes à étudier.

Une fois le corpus réuni dans les répertoires et ces répertoires indiqués comme ci-dessus dans le fichier descriteur de corpus, la construction du corpus est finie. Il est possible à tout moment de modifier la constitution du corpus en modifiant le descripteur de corpus et en indiquant dans le fichier de requête le nom du nouveau corpus.

E.3. Corpus balisé

Par défaut, c'est à dire si dans la requête il n'y a aucune indication de prise en compte des balises (nous expliquons cela dans les lignes qui suivent), SAGACE ne prend pas en compte les balises. Autrement dit et toujours sous réserve que l'option de prise en compte des balises n'ait pas été activée, celles-ci sont traitées comme du texte ordinaire.

Depuis la version 3.2, SAGACE peut traiter des textes balisés. Les balises, indiquées ci-dessous, portent sur la nature et la structure du texte.

E.3.1. Balises de délimitation d'un texte à l'intérieur d'un fichier <texte titre="..." date:"...">...</texte>

Un fichier peut être construit par concaténation de plusieurs textes indépendants. Il est alors possible de délimiter les textes à l'intérieur du fichier. C'est ce à quoi servent les balises <texte titre="..." date="..."> qui marquent un début de texte et </texte> qui en marque la fin.

Ces balises sont exploitées par les commandes "`Affiche le nom du texte original`" (voir chap.G.2.2.2, p.25) et "`Affiche la date du fichier original`" (voir chap.G.2.2.2, p.25).

E.3.2. Balises de titrage <TITRE rang="X"> ... </TITRE> (1 ≤ X ≤ 9)

Les titres sont balisés par <TITRE>. Toute balise de titrage doit comporter le niveau de titrage (valeur X) sous peine de ne pas être correctement interprétée par SAGACE.

Le niveau de titrage va de 1 à 9.

Le corps du texte, c'est-à-dire tout ce qui n'est pas balisé comme titre, n'a pas à être balisé. Tout segment

du corps est considéré par SAGACE comme balisé par défaut balisé <TITRE rang="10">. Il n'y a pas besoin de faire apparaître ces balises.

E.4. Bon à savoir

- Dans une recherche d'exemples, une option dans le fichier de requête (`Affiche le nom du fichier d'origine`, chap.G.2.2.2, p.24) permet d'enregistrer le nom du fichier dont est extrait l'exemple. Il est alors plus facile de retrouver le fichier si nécessaire.

- L'ouverture et la fermeture des fichiers sont autant d'actions qui prennent du temps lors de la recherche. Il y a tout intérêt à créer des gros fichiers pour éviter au logiciel d'ouvrir et de fermer une multitude de petits fichiers.

- Où trouver des textes numérisés libres de droit ? L'internet est une bonne ressource mais il faut prêter attention au cas par cas aux droits de copie et d'exploitation. Si l'auteur du site l'autorise, télécharger les pages avec texte (extension `txt`, `html`, `pdf`) etc, puis les transformer en fichiers textes (sans quoi les balises du texte original vont sensiblement affecter les résultats des manipulations avec SAGACE).

- Il est possible d'effectuer des recherches à "plusieurs étages", en exploitant à titre de corpus des recherches antérieures. C'est intéressant lorsque dans un corpus très volumineux, on veut étudier plusieurs distributions pour un morphe donné. Si la recherche est relancée à chaque fois sur tout le corpus, l'ensemble du travail va devenir particulièrement laborieux. Pour réduire sensiblement le temps de travail, une technique consiste à extraire tous les segments de textes contenant le morphe-lemme qui intéresse l'utilisateur, et à limiter ensuite les recherches à ces segments. Pour cela, il suffit de procéder comme dans l'exemple ci-dessous.

E.5. Exemple de création de corpus *ad hoc* pour accélérer une recherche

Mettons que nous voulions étudier trois distributions (DISTRIB1,DISTRIB2 et DISTRIB3) données pour un morphe MORPHE, dans un corpus CORPUS1 de plusieurs millions de phrases. La démarche la plus simple consiste à lancer trois recherches sur CORPUS1. Cela signifie que SAGACE va parcourir trois fois le CORPUS1, ce qui peut prendre un temps considérable.

Pour réduire sensiblement le temps de travail, nous extrayons du CORPUS1 un corpus plus court (CORPUS2) qui ne réunit que les segments contenant MORPHE. La recherche des trois distributions sera menée sur CORPUS2.

Voici les réglages à effectuer.

1) Lancer la recherche sur CORPUS1 en donnant à ce corpus un nom que nous indiquons dans le fichier de requête, et dont la localisation est donnée dans le descripteur de corpus :

Fichier de requête :

```
Groupe:corpus1
```

Descripteur de corpus :

```
groupe:corpus1
/home/truc/.../corpus1/
```

Le fichier de résultat FRESU1 sera enregistré dans un répertoire REP1 (par exemple : `/home/truc/.../corpus2/resu1.txt` . C'est à indiquer aussi dans le fichier de requête :

```
Resu:/home/truc/.../corpus2/resu1.txt
```

2) La recherche de la distribution DISTRIB1 sera menée sur le CORPUS2. Les réglages sont les suivants :

Fichier de requête :

```
Groupe:corpus2
```

Descripteur de corpus :

```
groupe:corpus2
/home/truc/.../corpus2/
```

3) La recherche des deux autres distributions se feront avec le même réglage qu'en (2).

F. Construire le lexique

Pour toute recherche de morphème par donnée de la catégorie, SAGACE travaille avec un lexique. Dans le lexique est décrit la catégorie (nom de catégorie et contenu).

Lorsqu'une position dans une chaîne à chercher est décrite par une catégorie syntaxique, SAGACE consulte le lexique et liste tous les morphes-lemmes qui appartiennent à cette catégorie. Par exemple, pour chercher une chaîne S en japonais de la forme " の + nom commun + の " où la deuxième position est occupée par un morphe-lemme de la catégorie des noms communs, SAGACE consulte le lexique et récupère tous les morphes-lemmes de la catégorie "nom commun". Lors de la recherche, SAGACE retiendra toutes les chaînes où la position de "nom commun" est occupé par un des morphe-lemmes du lexique.

SAGACE n'effectue pas d'analyse morphologique et ne calcule aucune dérivation ou flexions. En conséquence, il cherche dans le corpus exactement ce qui est dans le lexique. D'où l'appellation de morphe-lemme que nous désignerons désormais seulement par *lemme*.

Attention: l'encodage du lexique doit être identique à celui de la requête et du corpus

F.1. Compatibilité avec les version antérieures de SAGACE

Le moteur de SAGACE-v.3.X.X ne supporte plus les descriptions des propriétés syntaxiques utilisées dans les versions antérieures.

Dans la version 1.X.X, SAGACE utilisait un fichier décrivant la structure du lexique et son organisation mais ce dispositif est définitivement abandonné. SAGACE 3.X.X ne supporte plus ce dispositif.

F.2. Ce qu'est informatiquement le lexique

Le lexique est une collection de fichiers au format texte rassemblés sous une racine-répertoire unique. Le nombre et le nom des fichiers sont libres et sont répartis librement dans des sous-répertoires de la racine. Tous les fichiers présents sous cette racines sont considérés comme des fichiers du lexiques.

Lorsque SAGACE consulte le lexique, il le parcourt récursivement et explore tous les fichiers de tous les sous-répertoires sous la racine.

Lors de la conception du lexique, il est intéressant pour des questions pratiques de mettre une catégorie par fichier et de donner au fichier un nom explicite mais il n'y a en la matière aucune contrainte.

F.3. Localiser le lexique sur le système

Dans le fichier de requête, à la ligne « RepertoirLexic », indiquer le répertoire racine du lexique à exploiter :

```
RepertoirLexic:repertoire_racine_du_lexique
```

Par exemple, si tous les fichiers sont dans le répertoire `/home/truc/sagace/lexic` et ses sous-répertoires, renseigner la ligne ainsi :

```
RepertoirLexic:/home/truc/sagace/lexic
```

F.4. Rédaction des fichiers du lexique

Fondamentalement, le fichier est une liste de ligne composées du lemme et de sa représentation (les informations) entre crochets :

lemme [[représentation]]

Deux variantes existent : la liste sans aucune information, ou bien la liste avec information valable pour plusieurs lemmes.

F.4.1. Description syntaxique individuelle ("description locale")

Pour indiquer au niveau de chaque lemme sa représentation syntaxique (catégorie syntaxique ou toute autre information), mettre la représentation syntaxique à droite de chaque entrée :

$lemme_1$ [[$représentation\_syntaxique_1$]]
$lemme_2$ [[$représentation\_syntaxique_2$]]

...

Pour chaque entrée, SAGACE ajoute automatiquement les traits :

$lemme_i$ & lemme:$lemme_i$

Une entrée lexicale est donc interprétée par SAGACE :

$lemme_i$ [[représentation_syntaxique$_i$ & $lemme_i$ & lemme:$lemme_i$]]

Par exemple, l'entrée :

私 [[pronom & personne:singulier & traduction:français:je]]

est interprétée par SAGACE :

私 [[pronom & personne:singulier & traduction:français:je & 私 & lemme:私]]

La forme du lemme est laissée à l'utilisateur. Il ne doit pas contenir les sous-chaînes [[ni]] et ne doit pas commencer par la chaîne "cat:" .

Il y a un lemme par ligne.

Le lemme figure en début de ligne (non précédé d'un espace ou autre).

Améliorations envisageables : accepter dans le lemme la présence de [[et]]

F.4.2. Liste simple de lemmes

La forme la plus simple de rédaction est une liste de lemmes sans aucune indication :

lemme$_1$

lemme$_2$

...

Chaque lemme se verra attribuer automatiquement par SAGACE la représentation :

lemme$_1$ & lemme:*lemme$_i$*

lemme$_2$ & lemme:*lemme$_2$*

En définitive, les listes sont interprétées par SAGACE :

lemme [[*lemme$_1$* & lemme:*lemme$_i$*]]

Par exemple, la liste

私

あなた

sera interprétée :

私 [[私 & lemme:私]]

あなた [[あなた & lemme:あなた]]

F.4.3. Description syntaxique globale

Lorsque dans un fichier, plusieurs lemmes partagent les mêmes propriétés/traits syntaxiques, il est pratique de recourrir à une description globale qui épargne d'avoir à répéter la même information au niveau des entrées. Voici la syntaxe.

```
fichier du lexique
cat: représentationGlobale1
lemme11 [[représentation_locale11]]
lemme12 [[représentation_locale12]]
...

cat: représentationGlobale2
lemme21 [[représentation_locale21]]
lemme22 [[représentation_locale22]]
...
```

La ligne commençant par le mot réservé "`cat:`" n'est pas un lemme. La représentation syntaxique qui suit le mot réservé "`cat:`" vaut pour tous les lemmes qui suivent, jusqu'à la prochaine occurrence d'une ligne commençant par le mot réservé "`cat:`".

La représentation syntaxique d'un $lemme_i$ vaut la conjonction de la représentation globale, de la représentation locale et de "$lemme_i$ `&` `ecriture:val:`$lemme_i$.

Ainsi, la représentation syntaxique de $lemme_{12}$ vaut :

$représentationGlobale_1$ `&` $représentation\_locale_{12}$ `&` $lemme_{12}$ `&` `ecriture:val:`$lemme_{12}$.

Exemple :

Le fichier `pronom.dic` peut aussi être rédigé de la façon suivante :

pronom.dic

```
cat: nom & pronom
私    [[traduction:francais:val:je]]
あなた [[traduction:francais:val:tu]]
...
```

Lorsqu'il extrait les lemmes cherchés, SAGACE ajoute automatiquement le lemme lui-même à la formule de traits, de sorte que le lemme apparaît aussi comme un trait. Ainsi le fichier `pronom.dic` ci-dessus est en fait "lu" par SAGACE comme suit :

pronom.dic

```
cat: nom & pronom
私    [[traduction:francais:val:je & 私 ]]
あなた [[traduction:francais:val:tu & あなた]]
...
```

F.4.4. Commentaires dans un fichier

La double barre // commente toute la partie de ligne située à sa droite. La partie commentée n'est pas prise en compte lors de la consultation du lexique.

Exemple :

`太郎  // c'est le nom de mon grand-père`	Seul 太郎 est pris en compte
`// 太郎  c'est le nom de mon grand-père`	Rien n'est pris en compte.

Pour commenter plusieurs lignes contigües, les insérer entre deux lignes, la première comportant la balise <commentaire>, la seconde la balise </commentaire>. Par exemple :

Exemple:

```
<commentaire>
première ligne inutile blahblah
deuxième ligne inutile
</commentaire>
```

F.5. Représentation syntaxique du lemme

La description syntaxique est rédigée avec des formules propositionnelles de traits (voir chap.I,p.40).

F.5.1. Représentation syntaxique du lemme, sans contrainte sur l'environnement

Sont exclus de la représentation les traits distrib:herite:X et distrib:transmet:X, quelle que soit la valeur de X.

F.5.2. Représentation des contraintes imposées par le lemme sur son environnement dans le texte

SAGACE peut tenir compte de contrainte sur la distribution immédiate d'un lemme.

Par exemple, en japonais, le choix d'un morphe de conjugaison pour un radical/morphe donné dépend de la catégorie du radical. Ainsi le morphe de conjugaison んで est compatible exclusivement avec les radicaux verbaux de la catégorie dite « en n- ou m- ». Un tel radical n'est pas compatible avec le morphe-conjugaison って. En l'absence d'une analyse syntaxique et sémantique, pour limiter les risques d'erreurs lors de la recherche, un contrôle de combinaison à lieu, si celui-ci a été signalé dans le lexique. Ainsi, il est possible de contrôler la combinaison de sorte que dans la construction 会飲って, SAGACE ne considérera pas la chaîne 飲って comme la construction conjuguée vradical+conjugaison en って car le vRad 飲 est incompatible avec le morphe って. La seule construction acceptée est 飲んで.

Cette technique simple permet de limiter quelque peu le nombre d'erreurs lors de la recherche.

Pour indiquer qu'une catégorie pose une contrainte sur le mot qui suit, ou obéit à une contrainte imposée par le mot qui précède, ou les deux en même temps, on ajoute dans la représentation sémantique respectivement les traits `distrib:transmet:`*X* et `distrib:herite:`*X* avec *X* une valeur donnée par l'utilisateur. Cette valeur est un unique caractère, chiffre ou lettre.

Attention : La notation de la forme `[(herite-)principal(-transmet)]`n'est plus supportée à partir de la version 3.2.X.

Ainsi schématiquement les entrées se présentent :

```
lemme  [[... & distrib:herite:X & ... ]]
...
```

```
lemme  [[... & distrib:transmet:X & ...]]
...
```

```
lemme   [[... & distrib:herite:X & distrib:transmet:Y & ...]]
...
```

SAGACE gère la contrainte ainsi : un lemme de type `distrib:transmet:`*c* n'est reconnu dans un texte que si il est immédiatement suivi d'un lemme de type `distrib:herite:`*c*, et vice versa.

Exemple :

Considérons le lexique suivant :

```
cat: vRad
お呼び      [[distrib:transmet:3]]
お目にかか   [[distrib:transmet:r]]
お          [[distrib:transmet:k]]
お          [[distrib:transmet:r]]
お会い      [[distrib:transmet:3]]
お越し      [[distrib:transmet:3]]
...
かぶれ      [[distrib:transmet:1]]
かぶ        [[distrib:transmet:r]]
かまけ      [[distrib:transmet:1]]
...

cat: conjTa
た       [[distrib:herite:1]]
た      [[distrib:herite:5]]
った     [[distrib:herite:w]]
いた     [[distrib:herite:k]]
った     [[distrib:herite:7]]
いだ     [[distrib:herite:g]]
した     [[distrib:herite:s]]
じた     [[distrib:herite:z]]
った     [[distrib:herite:t]]
んだ     [[distrib:herite:n]]
んだ     [[distrib:herite:b]]
んだ     [[distrib:herite:m]]
った     [[distrib:herite:r]]
```

```
した     [[distrib:herite:3]]
かった    [[distrib:herite:i]]
```

Dans le corpus, l'occurrence de お呼び ne peut être reconnu comme le lemme ci-dessus que si il est suivi d'un lemme, de type [distrib:herite:...] de contrainte gauche égale à 3. Par exemple, la chaîne お呼びした est reconnue par SAGACE comme une chaîne de la forme "vRad-conjTa", tandis que par exemple かぶした sera exclue puisque かぶ doit être suivi d'un lemme marqué r, ce qui n'est pas le cas de した.

F.5.3. Consultation en ligne du lexique

SAGACE dispose de plusieurs commandes qui permettent de consulter rapidement un lexique pour en connaître le nombre d'entrées, pour lister les traits et les catégories. Voir le chapitre “ligne de commandes” (chap.K). Ces commande sont pratiques lors de la conception du lexique.

F.5.4. Construire des catégories à la volée

Le langage LEXSPROP permet de construire des nouvelles catégories "à la volée" par simple jeu sur les traits. Voir la description du fichier de requête, chap.I.

F.5.5. Bon à savoir

- Il est possible de produire de nouvelles catégories à partir de résultats antérieurs. Pour cela, il suffit de faire une recherche de type "listage de collocation" en n'enregistrant qu'un seul lemme, de récupérer la liste ainsi produite, de lui attribuer une catégorie et de mettre le fichier dans le lexique.

- SAGACE ne se soucie pas de l'environnement des chaînes qu'il cherche. Ainsi, lors d'une recherche d'occurrence de la particule から, SAGACE acceptera une occurrence de から, même si cette chaîne est immédiatement suivie de すを. Or très vraisemblablement, dans une chaîne ...からすを..., から n'est nullement la particule casuelle, mais la première partie du nom commun "からす" suivi de la particule casuelle を. Il est conseillé de relever dans chaque langue les symboles, morphes et autres chaînes qui n'apparaissent dans aucun lemme du lexique, ou du moins que très rarement. Typiquement pour le japonais, la particule を est un marqueur intéressant car elle n'apparaît dans aucun autre mot. Il en va de même de la ponctuation. On peut alors se servir de ces caractères pour désambiguïser les chaînes à chercher (voir exemple dans le tutoriel, chap.L.5, p.50).

F.6. Insérer des entrées *via* la commande en ligne

Il est possible d'intérer de nouvelles entrées dans le lexique via la commande en ligne. Les nouvelles entrées seront insérées dans un fichier à part `ajout.dic`, qui sera localisé dans un répertoire `ajout` au niveau de la racine du lexique.

SAGACE crée automatiquement le fichier si celui-ci n'existe pas. Il faut que le sous-répertoire ajout existe et soit accessible en écriture et lecture.

Si le répertoire et le fichier existent déjà, SAGACE ouvrira et exloitera ceux là. Il faut que le répertoire et le fichier soient accessiblent en lecture et écriture.

Pour empêcher toute insertion, il suffit de ne pas avoir de sous-répertoire `ajout` ou de le rendre innaccessible en écriture par SAGACE.

Cette fonction a plusieurs intérêts.

- Procédure plus simple pour un accès très ponctuel au lexique.

- Il est possible de donner accès mais avec des restrictions d'écriture. Le cas le plus fréquent sera certainement celui qui consiste à interdire en écriture les données existantes et de ne donner accès en écriture qu'au répertoire `ajout`.

La ligne de commande se présente :

```
sagace [-f cheminNomFIchierReqet] --inserBasicLexic "entree [[liste des traits]]"
```

L'entrée peut être sans représentation syntaxique. Celle-ci sera alors celle attribuée par défaut par SAGACE, si le lexique est exploité par SAGACE.

Les guillemets sont obligatoires si la chaîne entrée contient des espaces. Si la chaîne entrée contient des

guillemets, il faut les faire précéder d'une barre inversée :

```
ecriture:val:\"日本語\"
```

Attention : ce mode d'insertion ne fait aucun contrôle de cohérence entre les nouvelles données et celles existantes.

Améliorations envisageables : contrôler la cohérence entre les données.

G. Paramétrage de la requête pour un texte quelconque

L'explication donnée dans les lignes qui suivent concerne une recherche sans prise en compte de la structure du texte (distinction entre titre et corps de texte). La gestion de la hiérarchie est expliquée au chapitre H (p.39).

Le paramétrage d'une recherche se fait dans un fichier texte dit "fichier de requête".

Sont notifiés dans ce fichier les paramètres de la recherche (type de recherche, emplacement du descripteur du corpus, emplacement du lexique, etc...) et la chaîne de lemmes à chercher. Typiquement, le fichier se présente comme suit:

Structure	Exemple

En-tête :
Passage des paramètres de la requête
 	
(notifications optionnelles)
 	
Chaîne recherchée
	`Travail: exemples`
	`Resu:/home/truc/resu/truc.dat`
	`RepertoirLexic:/home/truc/dico/dicoJaponais/`
	`ScriptCorpus:/home/truc/txt/txtJaponais/scriptCorpus.txt`
	`NomDuGroupe:japonais`
	`Arret:"。 "`
	`Langue: japonais`
	`Affiche le nom du fichier d'origine`
	`Taille max des exemples:50`
	`Nombre max des exemples:10`
	`Nombre max de cas dans un segment:0`
	`Taille max du fichier de resultats:210`
	`Elimine repetitions dans corpus:non`
	`Reqet:`
	
	`>0 等`
	`=0 cat:specNum /-affich:trait:lemme`
	`...`

Les options, rédigées immédiatement après une barre oblique unique et un tiret «/-» commandent les modes d'affichages des résultats dans le fichier de résultats.

La requête donnée en exemple ci-dessus indique que

(données obligatoires)

- le travail est une recherche d'exemples
- Le résultat est enregistré dans `/home/truc/resu/truc.dat`
- Le répertoire du lexique est `/home/truc/ dico/dicoJaponais/`
- Le descripteur de corpus est `/home/truc/txt/txtJaponais/scriptCorpus.txt`
- Le corpus est celui du groupe appelé « japonais »
- L'unité de recherche est le segment de texte délimité par deux ronds " 。 " (le second étant intégré au segment).

(données optionnelles)

- Les segments étudiés sont d'une longueur maximale de 50 octets
- Le fichier de résultats ne comprendra pas plus de 10 cas
- Nombre max de cas dans un segment: pas de limite maximum
- Le fichier de résultat ne dépassera pas la taille de 210 octets
- Les segments répétés sont tous pris en compte

La chaine recherchée :

- débute par le morphe 等, suivi immédiatement d'un spécifique numéral quelconque.

Les options d'afffichages des chaînes cherchées sont les suivantes :

- Il est demandé d'isoler le spécifique numéral trouvé dans cette chaîne (de sorte à être facilement visualisable.

Les chapitres qui suivent sont consacrés à la rédaction du fichier de requête.

G.1. Choix du fichier de requêtes

Le fichier de requête est au choix

- le fichier `reqet.txt` par défaut fourni avec SAGACE et placé dans le répertoire principal avec l'exécutable,

- un fichier placé n'importe où sur le système.

En l'absence d'indication dans la ligne de commande, SAGACE exploite le fichier par défaut. Pour utiliser un autre fichier, utiliser la commande `-f` (voir le chap.K, p.44).

G.2. Rédaction du fichier de requêtes

Le fichier se présente en deux parties. Dans la première partie, l'entête, figurent des données à renseigner obligatoirement sur l'environnement de SAGACE, sans lesqelles le logiciel ne peut fonctionner, ainsi que des données optionnelles correspondant plutôt à des restrictions de fonctionnement. La seconde partie du fichier contient la description de la chaîne à chercher.

G.2.1. Contrôle de la syntaxe du fichier de requêtes

(section concernant plutôt l'implémentation).

A partir de la version 3.2.0, un contrôle de la bonne rédaction du fichier de requête a préalablement lieu. Cette mesure permet avant tout, en isolant la procédure de contrôle, de séparer le contrôle et l'interprétation, de sorte que la rédaction du sous programme d'interprétation s'en trouve allégée.

Tous les contrôles ne sont pas systématiquement effectués à ce niveau. On a laissé en l'état les contrôles mêlés au programme d'interprétation et qui ne méritait pas une réécriture du programme.

Les notifications d'erreurs sont affichées dans le fichier de résultats. Elles sont aussi affichées sur la sortie écran si l'option `-verbose` est activée.

Implémentation dans le fichier : `verif_Syntax_fReqet.c`

G.2.2. Entête du fichier de requête

Il commence par les données obligatoires et finit par une ligne "Reqet".

G.2.2.1 Entête ; données obligatoires

Voici une présentation générale des données à renseigner obligatoirement pour que SAGACE fonctionne.

(en italique les valeurs laissées à l'utilisateur)

Rubriques	valeurs possibles
`Travail:`	`exemple`
	`exemple+devant`
	`stat`
`RepertoirLexic:`	*chemin*
`ScriptCorpus:`	*chemin et nom*
`NomDuGroupe:`	*nom du groupe*
`Arret:`	"*chaîne*$_1$" "*chaîne*$_2$"...

La rubrique doit obligatoirement être écrite, en tête de ligne (pas d'espace devant ou autre).

A partir de la version 4.2.0, il n'y a plus aucune spécificité propre à une langue. Cette option n'est plus disponible.

1) `Travail:`

Trois valeurs possibles : exemples | exemples+devant | stat
Avec cette rubrique l'utilisateur indique le type de recherche à effectuer :
exemple : recherche de segments de texte contenant la chaîne indiquée.
exemple+devant : idem mais en affichant en plus le segment qui précède.
stat : listage et éventuellement comptage des collocations

2) `RepertoirLexic:`
Valeurs : chemin complet du lexique à exploiter (voir chap.F.3, p.16).

3) `ScriptCorpus:`
Valeurs : chemin complet et nom du descripteur du corpus (voir chap.E.2,p.13).

4) `NomDuGroupe:`
Valeurs : un des noms de corpus parmi ceux donnés dans le descripteur de corpus.

5) `Arret:`
Valeur : une suite de mots chacun entre guillemets ou `saut_a_la_ligne.`
Chaînes marquant la fin des segments de travail.
A partir de la version 3-4, il est possible de proposer plusieurs marques de fin de segment. Par exemple : “。” ”!” “?” . En général, on travaillera sur la phrase, laquelle se termine en japonais par "。" et "." en coréen. Mais rien n'interdit de se concentrer sur d'autres unités, comme les paragraphes (ce qui suppose d'insérer dans le texte des marques de paragraphe). Si par exemple le corpus est balisé et le paragraphe borné par les balises et , la rubrique "Arret:" prendra la valeur .
Il est possible de donner comme arrêt le "saut à la ligne". Sous linux, il s'agit du caractère '\n'. Pour cela, indiquer sans guillemets et sans accent sur le "a". Dans de nombreux textes, le saut à la ligne correspond à une fin de paragraphe.

G.2.2.2 Paramétrage de la requête - entête ; données optionnelles

Plusieurs options sont valuées par défaut si l'utilisateur ne les mentionne pas explicitement. Elles concernent essentiellement des limitations sur la recherche et sur l'enregistrement des résultats.

Voici ces options:

	Intitulé du paramètre optionnel	Valeur par défaut	Valeurs possibles
1	Affiche le nom du fichier d'origine	non	oui si affiché
2	Affiche le nom du texte original	non	oui si affiché
3	Affiche la date du fichier original	non	oui si affiché
4	Affiche la src du fichier original	Non	Oui si affiché
5	Taille max des exemples:	1024	donner un nombre
6	Nombre max des exemples:	2 milliards	donner un nombre
7	Taille max du fichier de resultats:	2 milliards d'octets	donner un nombre
8	Elimine repetitions dans corpus:	non	non globalement par_fichier
9	Sauvegarde automatique:	non	oui si affiché
10	Sauvegarde au format KWIC [sans repetition] **[avant=*X*] [apres=*Y*]**	non	oui/non
11	Afficher le KWIC en HTML: [“*nom_du_fichier*”]	non	oui si affiché
12	Numerote les segments	non	oui si affiché
13	Encodage:*X*	Utf8	*X*
14	Stat ordonnee:X	non	X vaut : non (si non affiché) croissant decroissant
15	Convertit les caracteres latins en 1 octet	Non	Oui si affiché sans ‘non’
16	Erreur syntaxe lexiqe	Aucune action	Signale ET/OU stop
17	Nombre max de cas dans un segment: ATTENTION:BUG DANS LA COMMANDE QI NE DEMARRE Q’A PARTIR DE 2COMPTE ALORS 1. CERTAINEMENT UN FOR I=1;TO I<1	0	0 : pas de limite 1, 2, ... : nombre max
18	Backtrack sur position:	apres	+1
19	InsereDebSegment:"X"	""	X

1) `Affiche le nom du fichier d'origine`

Valeurs possibles : oui | non . Valeur par défaut : non.

Option opérationnelle uniquement pour la fonction de concordancier

Concerne l'affichage des résultats.

Dans le fichier de résultats, affiche en tête du segment extrait le nom du fichier dans lequel l'occurrence a été trouvée. Par exemple

```
::>
waganeko : 我が猫だ。
```

En l'absence de cette option (ou si cette option est valuée "non"), le résultat se présentera :

```
::>
我が猫だ。
```

Dans un relevé de collocations, l'activation de cette option n'est pas prise en compte mais contrairement aux versions précédentes de SAGACE, elle ne provoque pas l'arrêt de la recherche.

2) `Affiche le nom du texte original`

Valeurs possibles : oui | non . Valeur par défaut : non.

Ne fonctionne que si le texte étudié contient les balises <texte > (voir chap.E.3, p.14). Affiche alors le nom du texte indiqué dans la balise <texte titre="..." ...> dont la portée contient le segment étudié.

Cette fonction ne s'assure pas que le texte est correctement balisé. Elle ne contrôle pas la présence de la balise </texte>.

3) `Affiche la date du fichier original`

Valeurs possibles : oui | non . Valeur par défaut : non.

Ne fonctionne que si le texte étudié contient les balises <texte > (voir chap.E.3, p.14). Affiche alors le nom du texte indiqué dans la balise <texte date="..." ...> dont la portée contient le segment étudié.

Cette fonction ne s'assure pas que le texte est correctement balisé. Elle ne contrôle pas la présence de la balise </texte>.

3) `Affiche la src du fichier original`

Valeurs possibles : oui | non . Valeur par défaut : non.

Affiche l'attribut src présent dans la balise <texte ... >.

Cette fonction ne s'assure pas que le texte est correctement balisé. Elle ne contrôle pas la présence de la balise </texte>.

5) `Taille max des exemples:`

Valeurs possibles : nombre d'octets quelconque . Valeur par défaut : 1024 octets.

Restreint la recherche.

Cette option restreint la recherche aux segments ayant la longueur maximum indiquée. Cette longueur est donnée en octets. Avec un encodage de caractères sur 2 octets, il faut diviser la longueur indiquée par deux pour obtenir le nombre de caractères.

6) `Nombre max des exemples:`

Valeur possible : valeur quelconque. Valeur par défaut : aucune

Restreint la recherche.

Ne concerne que la recherche d'exemples.

Cette option provoque l'arrêt de la recherche une fois que le nombre de segments trouvés a atteint la limite indiquée. En l'absence d'un chiffre maximum, il n'y a aucune limite et la recherche ne s'arrête que lorsque le corpus a été complètement parcouru.

7) `Taille max du fichier de resultats:`

Valeurs possibles : nombre d'octets quelconque. Valeur par défaut : aucune.

Restreint la recherche.

Ne concerne que la recherche d'exemples.

Cette option provoque l'arrêt de la recherche une fois que la taille du fichier de résultats (dans lequels les segments trouvés sont enregistrés) a atteint la taille indiquée. En l'absence d'un chiffre maximum, il n'y a aucune limite et la recherche ne s'arrête que lorsque le corpus a été entièrement parcouru.

8) Elimine repetitions dans corpus:

Valeurs possibles : non | globalement | par_fichier . Valeur par défaut : non

Restreint la recherche.

Concerne tous les types de recherches

Si cette option est activée, SAGACE ne travaille que sur la première occurrence d'un segment. Les autres occurrences ne seront pas prises en compte. Voici l'effet obtenu selon les valeurs adoptées :

globalement : le contrôle est effectué sur la totalité du corpus. Les répétitions seront éliminées même si elles n'apparaissent pas dans le même fichier que la première occurrence.

par_fichier : le contrôle est effectué à l'intérieur de chaque fichier. Deux occurrences dans deux fichiers différents ne sont pas considérées comme des répétitions.

Cette option doit être activée avec précaution. Elle entraîne en effet un ralentissement très sensible de la recherche et une consomation considérable de mémoire. Si cette option doit absolument être activée, la valeur "par_fichier" est un moindre mal. Autrement, il est certainement préférable de débarrasser le corpus de ses répétitions avant même de le soumettre à SAGACE.

9) Elimine repetitions dans corpus

Valeurs possibles : oui | non . Valeur par défaut : non

Concerne le listage de collocations.

Lors d'un listage de collocations (avec ou sans comptage), par défauts, les résultats sont enregistrés dans la mémoire vive. Ils ne sont enregistrés dans le fichier final qu'à la fin de la recherche. En cas d'arrête inopiné du logiciel (par exemple un ordinateur qui plante), les résultats obtenus avant l'arrêt sont perdus. Cela peut être préjudiciable si la recherche prend du temps.

Il est donc possible d'activer une sauvegarde par défaut qui à intervales réguliers sauvegardera les résultats en cours. Cette sauvegarde ne tient pas compte de la taille du résultat à sauvegarder. Elle a lieu même si le résultat est nul.

La première sauvegarde est déclenchée après analyse de 50 segments, les suivantes le sont à n+10 avec n la valeur de déclenchement au cycle précédent.

La sauvegarde automatique ralentit la recherche. Elle n'est donc conseillée qu'en cas de nécessité.

Voici un exemple :

```
Affiche le nom du fichier d'origine
Nombre max exemples:205
Taille max des exemples:404
Taille max du fichier de resultats:1000000
Elimine repetitions dans corpus:globalement
Reqet:
```

La présence de "Reqet" en fin d'entête est obligatoire.

10) Sauvegarde au format KWIC [sans repetition][avant=*X*] [apres=*Y*]:

Valeurs possibles : oui | non . Valeur par défaut : non

Concerne la fonction de concordancier.

Cette fonction rend inopérante la fonction "phrase précédente" ainsi que les options d'affichage indiquées au niveau des composants de la chaîne à chercher.

Les chaînes trouvées sont affichées au format KWIC (Key Word In Context) , en donnant *X octets*[2] avant et *Y octets* après. Par défaut, *X* vaut 20 et *Y* vaut 20 octets.

Voici deux affichages d'exemple, l'un au format brut (non KWIC) :

```
::> 三
phraseEtudiee: 學變臆説
内藤湖南

------------------------------------------------------
【テキスト中に現れる記号について】
```

2 Prêter attention au fait qu'il s'agit d'octets et non de caractères.

```
［#］：入力者注　主に外字の説明や、傍点の位置の指定
      （数字は、JIS X 0213の面区点番号、底本のページと行数）
（例）［#「土へん＋侯」、読みは「こう」、第4水準2-5-1、352-2］
-------------------------------------------------------

天運は循環するか、意ふに其の循る所の環は、完全なる圓環にあらずして、寧ろ無窮なる螺旋形を爲す者たらんか、何となれば一個の中心點より開展して三のダイメンシヨンある空間を填充すべき一條線は、須らく無限に支派して螺形に纏繞する所の者たらざるべからざれば也。

::> 一

phraseEtudiee: 藏書家の話
内藤湖南

-------------------------------------------------------
【テキスト中に現れる記号について】

［#］：入力者注　主に外字の説明や、傍点の位置の指定
（例）（昭和二年十一月「書物の趣味」第一册）［#地付き］

／＼：二倍の踊り字（「く」を縦に長くしたような形の繰り返し記号）
（例）だん／＼増補して、
-------------------------------------------------------

　清朝はその初頃から有名な藏書家が多く、錢謙益及その族孫錢曾、又は季振宜などは、順治より康熙の初年に有名であるが、併し藏書家の最盛期は乾隆の中頃以後にあるので、乾隆の末から嘉慶を經て、道光の初頃まで居つた蘇州の黄丕烈は最も有名で、殆ど清朝を通じて第一の藏書家と言つてよいのである。

::> 四
phraseEtudiee: 水仙
太宰治
「忠直卿行状記」という小説を読んだのは、僕が十三か、四のときの事で、それっきり再読の機会を得なかったが、あの一篇の筋書だけは、二十年後のいまもなお、忘れずに記憶している。
```

et le même au format KWIC :

```
[なれば一個の中心點より開展して　/　三　/　のダイメンシヨンある空間を填充]
[最も有名で、殆ど清朝を通じて第　/　一　/　の藏書家と言つてよいのであ
る。]
[小説を読んだのは、僕が十三か、　/　四　/　のときの事で、それっきり再読
の]
[」が国民之友新年附録中に就て第　/　一　/　の傑作たるは世人の許す所な
り。]
[きおへないうちに、甚だ重要な二　/　三　/　の議論が私の眼にふれた。|]
```

Le format KWIC consiste en fait à aligner au milieu de la ligne les mots trouvés, de sorte que leur repérage soit immédiat.

L'option `sans repetition` permet d'éliminer les segments répétés et de n'enregistrer qu'une seule occurrence. Attention : il s'agt des segments répétés «complets» et non pas le segment kwic. De ce fait, si un patron cherché apparaît plusieurs fois dans la phrase, seule sa première occurrence sera exploitée. Ceci n'a pas de sens et doit être modifié à terme.

Voir G.2.3.3 (p.36)les options d'affichage des résultats pour décider du lemme de la chaîne cherchée qui sera affichée au centre.

Niveau implémentation : SAGACE n'enregistre pas directement le fichier dans ce format. Il crée d'abords un fichier ordinaire temporaire à partir duquel il produit un second fichier, au format KWIC. Dans le premier fichier, à chaque chaîne à sauver, enregiste dans chainon->position la position du mot de référence. Une fois la recherche d'exemples finie, tous les segments sont extraits, tronquées selon X et Y, et enchaînés dans l'ordre alphabétique à partir de la position de référence, dans une chaine sans répétition. Puis le tout est enregistré dans le fichier de résultats. Le fichier temporaire est ensuite éliminé.

Le choix de faire ce tri après la recherche plutôt que pendant permet de ne pas alourdir la recherche elle-même et d'avoir un fichier de secours au cas ou le programme planterait en cours d'enchaînement. La

contrepartie est la multiplication des accès au disque et un problème éventuellement de gestion des droits si ceux-ci sont limités.

11) `Afficher le KWIC en HTML: [“nom_du_fichier”]`

Valeurs possibles : oui | non . Valeur par défaut : non

Si “nom_de_fichier” est absent (ie est de longueur nulle), c’est la feuille par défaut qui est prise en compte.

Le nom du fichier ne doit pas faire plus de 200 caractères

Le résultat au format KWIC est donné dans un fichier HTML. Il faut indiquer une feuille de style. Par défaut, c’est la feuille “style.css.txt” présente dans le répertoire par défaut (celui où se trouve l’exécutable sagace qui est pris en compte.

La feuille de style n’en est pas une au sens stricte du terme. En fait elle contient seulement l’entête html qui figurera en tête du fichier de résultat. Dans cette entête apparaissent la définition des styles.

Il est possible à l’utilisateur de fixer ses propres styles. Dans ce cas, il doit impérativement définir

table : le style du tableau

td.c1 : première colonne, contenant le numéro du l’exemple extrait,

td.c2 : deuxième colonne, contient l’environnement gauche de la référence

td.c3 : troisième colonne, convient la référence

td.c4 : quatrième colonne, contient l’environnement droite de la référence

td.c5 : cinquième colonne, contient des données diverses (nom de fichier, de texte etc.)

td.c0 : colonne ad hoc encadrant le référent

L’entête est insérée dans le fichier résultat dont le <body> contient les résultats.

Attention : il faut adapter l’encodage des caractères à celui utilisé dans la requête, le corpus et le lexique. Le format par défaut est sjis :

```
<meta http-equiv="content-type" content="text/html; charset=sjis">
```

Implémentation : l’option est gérée avec option.fichierCSSprKWIC . Si l’option est désactivée, le fichier est NULL. Si aucun nom de fichier n’est (correctement) indiqué, par défaut prend le fichier style.css.txt dans le répertoire courant (et suppose que sagace est lancé en ligne à partir du répertoire courant. Sinon, prend le répertoire fichier indiqué entre guillemets.

12) `Numerote les segments`

Valeurs possibles : oui | non . Valeur par défaut : non

Concerne la fonction de concordancier.

Numérote les exemples trouvés.

13) `Encodage:X`

Valeurs possibles : *X* . Valeur par défaut : utf8

Cette fonction n’est utile que pour le japonais, lorsque la “sauvegarde sous la forme conjuguée” est activée.

Si il est demandé de présenter les radicaux verbaux sous leur forme conjuguée, SAGACE va ajouter la conjugaison. Pour le faire correctement, SAGACE a besoin de connaître l’enocodage du fichier de résultats. Par défaut considère que c’est de l’utf8.

Deux encodages sont supportés, utf8 et sjis.

14) `Stat ordonnee:X`

Valeurs possibles : `non` | `croissant` | `decroissant` . Valeur par défaut : `non`

La valeur est nulle (si l’option n’est pas affichée). Si l’option est affichée sans valeur pour X, c’est l’ordre décroissant par défaut. Sinon, la liste des collocations est ordonnée dans l’ordre croissant ou décroissant de la dernière valeur numérique.

Pour que cette option soit appliquée, il faut que l’un des éléments de la chaîne à chercher porte l’option `/-compteRef`.

Implémentation : l’ordonnancement ne se fait pas en direct mais dans seulement une fois la recherche

finie.

15) Convertit les caracteres latins en 1 octet

Valeurs possibles : non | oui

Transforme tous les caractères latins et chiffres du corpus codés sur 3 octets (utf8) en un caractère ou chiffre encodé sur un octet (codage ascii). Cette conversion a lieu à la volée, sur le segment extrait et avant recherche de la chaîne. Cette option n'a d'intérêt que pour le texte en utf8.

Cette option a été conçue et implémentée pour le japonais encodé en utf8. Elle devrait servir pour toute langue, sous réserve que l'encodage est de l'utf8.

Elle part du constat que les caractères japonais dans les corpus sont tantôt encodés sur 3 octets (par exemple " A " :<U0xef.U0xbc.U0xa1>), tantôt sur 1 octet ("A", <U0x41>). SAGACE les traite alors comme deux caractères différents. Pour les traiter comme un seul et même caractère dans les mots, il faut lexicaliser chaque lemme dans toutes les variantes d'encodage : " U N " et "UN". C'est très lourd, d'autant qu'on ne peut exclure des mélanges dans le corpus, qui ne pourront être pris en compte. Par ailleurs, au moins en japonais, la distinction n'est linguistiquement pas pertinente. Pour ne pas avoir à démultiplier les entrées lexicales avec les conséquences sur la manipulation du lexique et si on ne veut pas unifier toutes les variantes dans un prétraitement ue corpus, il reste cette option, qui effectue une conversion à la volée.

Implémentation : l'implémentation est très rudimentaire puisque chaque postion du corpus sera comparée aux lettres d'une valeur encodée dans un tableau. L'extraction des segments du corpus est donc ralentie si cette option est activée. Pour gagner du temps, il vaut mieux unifier l'encodage du corpus par un traitement préalable avant de lancer des recherches dessus avec SAGACE.

16) Erreur syntaxe lexiqe :

Valeurs possibles : signale , stop . Par défaut, aucune omission de l'entrée erronnée.

Gère les erreurs de rédaction des entrées dans le lexique.

Si une erreur est détectée dans la rédaction d'une entrée lexicale, l'entrée n'est pas prise en compte.

En mode "verbose" avec l'option signale, si une erreur est détectée.

L'option stop provoqe l'arrêt de la compilation du lexique lorsqu'une erreur est détectée.

17) Nombre max de cas dans un segment:

Valeurs possibles : 0 ou tout autre nombre. Par défaut, 0 (c'est à dire pas de limite maximum).

Permet de limiter le nombre de collocations trouvées dans un segment. Si le chiffre est fixé, cesse de chercher d'autres occurrences dans le segment en cours d'étude.

Cette option apparaît dans v4.2.0. Auparavant, les structures de 1 morphe étaient recherchées autant de fois qu'il y avait d'ocurrences. Les autres structures n'étaient dénombrées qu'une seule fois.

18) Backtrack sur position:

Valeurs possibles : +1 ou «apres».

Par défaut, «apres».

Lorsq'une reqête est achevée (sur un succès ou un échec), SAGACE cherche d'autres solutions pour chacun des composants. Soit il cherche sur la position du composant+1 soit sur la position apres celle qui était correcte pour le composant actuel.

Voici des exemples.

Soit en (1) un segment (une chaîne d'octets, chaque «o» représentant un octet) et entre crochets les composants du motif qui ont été trouvés jusqu'à cette étape. Si le deuxième composant est de position libre, on relancera la recherche d'un autre patron, dont le deuxième composant serait éventuellement sur une autre position. Cette nouvelle position (indiquée par un «+») sera sur l'octet suivant au cas où le backtrack est explicitement valué «+1» (ligne 2), soit qui débutera sur la position après la fin du composant actuellement reconnu dans tous les autres cas (ligne 3),

```
1)  o  o  o <o  o  o  o> o  o  o  o  o  o <o  o  o  o  o> o  o  o  o
2)  o  o  o <o  o  o  o> o  o  o  o  o  o  o  +  o  o  o  o  o  o  o
3)  o  o  o <o  o  o  o> o  o  o  o  o  o  o  o  o  o  o  +  o  o  o
```

Supposons cette fois-ci que le deuxième composant ne puisse plus «bouger» (soit parce qu'il est à position fixe par rapport au premier composant, soit parce que la nouvelle position le placerait à l'extérieur

du segment (nouvelle position > longueur du segment). Dans ce cas, il n'y a pas plus rien à tenter sur ce composant, le backtracking aura lieu sur le composant précédent. Si celui-ci n'est pas sur une position figée, alors sa possible nouvelle position sera de 1 octets plus loin (cas 4) ou bien immédiatement après la fin du premier composant (cas 5) :

```
4)  o o o o + o o o o o o o o o o o o o o o o
5)  o o o o o o o + o o o o o o o o o o o o o
```

18) InsereDebSegment:"X"

Valeurs possibles : X doit être de longueur inférieure à 100

Par défaut, X vaut "".

La chaîne X est insérée en tête de segment. Si rien n'est indiqué, X est une chaîne vide. Cette option répond au problème de la position du patron dans la phrase et à ses délimiteurs. Soit «cat:nom» le patron cherché. Supposons que l'on veuille borner ce patron, par exemple par «cat:d1» avant et après. Ce délimiteur n'est en principe pas nécessaire en tête de phrase, puisque la tête de phrase est en soi un délimiteur. Sans l'insertion d'un marqueur de début de phrase, pour chercher un patron comme ci-dessus, il faut en fait faire deux recherches : l'une du patron « cat:d1 cat:nom cat:d1 », et l'autre en début de phrase qui prendra la forme « cat:nom cat:d1 ». Pour éviter d'avoir ainsi à faire deux recherches, on peut insérer une marque de début de phrase, au choix, qui sera enregistrée en plus dans le lexique. Dans l'exemple, on intégrera naturellement la marque de début de phrase dans la catégorie d1.

Cette option est aussi tratique pour accélérer sensiblement le comptage des segments. Jusqu'à l'introduction de cette option, le comptage des segments se faisait en général en décomptant les marques de fin de segment, ce qui supposait de parcourir tout le segment. Désormais, il suffit de compter les marques de début de segments :

```
Travail: stat
...
Arret:"。"
...
InsereDebSegment:"X"

Reqet:
=0 X /-affich:"x" /-compte
```

G.2.3. Description de la chaîne recherchée

Une fois fixés tous les paramètres, commence la description de la chaîne à rechercher. Une requête porte sur une chaîne continue ou discontinue de morphes ou de classes de morphes. Par exemple, la recherche de la chaîne décrite comme suit :

```
>0 cat:numeral /-affich:trait:lemme
=0 cat:specNum /-affich:" " /-affich:trait:lemme
=0 の
>0 cat:vRad   /-affich:" " /-affich:trait:lemme
=0 cat:conjTa /-affich:trait:lemme
```

est une recherche n'importe où dans la phrase d'un *numéral*, suivi immédiatement d'un *spécifique numéral*, suivi immédiatement du morphe の, et suivi n'importe où après (contigü ou non) d'un *radical verbal* et d'une *conjugaison du parfait*. Il est en plus demandé que soient affichés les morphes trouvés (sauf の), en séparant le numéral du spécifique numéral, et le radical verbal. Ce qui produit:

```
::> 何 本 まとめた
shugiin0212.txt:  また、所信表明演説の中で、早急に総合的な対応策を取りまとめると述べていますが、あなたは今までに何本の対策方針ペーパーをまとめたのですか。

::> 一 層 求めた
shugiin0212.txt:  その意味で、深刻化している金融機関の貸し渋り、貸しはがし等をとめさせるとともに、政府系金融機関、信用保証協会によるセーフティーネット貸し付けや保証の思い切った拡充と、売掛債権担保融資制度の一層の推進、特別信用保証制度で延滞歴のない中小企業者
```

```
が追加保証を求めた場合の優遇措置等を実施すべきであります。

::> 一 国 差し伸べた
shugiin0212.txt:  痛みを耐えることができなかった人々の死に対して、あなたは、一国の総理として、どのような手を差し伸べたのでしょうか。
```

(les phrases sont extraites du site de la chambre des députés, relevé en décembre 2002). Attention, dans le fichier de résultat, les morphes trouvés ne sont pas colorés comme ici.

La même recherche sur les collocations produit :

```
...
万 両 記した
万 葉 きた
万 羽 舞った
万 羽 死んだ
万 羽 処分した
万 羽 なった
万 株 された
一 種 読んだ
一 種 作った
一 種 された
一 室 見つけた
一 部 いた
...
```

Dans la description, chaque ligne correspond à un composant de la chaîne cherchée. Elle est de la forme :

DISTANCE CATEGORE OPTION_AFFICHAGE
DISTANCE := = | >
NUM := [0..9]*
CAT := cat: | NULL
OPTION := affich:trait:ATOM:ATOM:...:ATOM | compte
SOPTION := (/-OPTION)*
COMPOSANT := DISTANCE NUM CAT F SOPTION

Par exemple :

>0 cat:(nom & particule) /-affich:trait:divers /-espaceDevant /-compte

G.2.3.1 Indice de proximité (> ou =) et valeur de proximité

> et = indiquent la position du morphe par rapport au précédent, ou par rapport au début de la phrase si il s'agit du premier.

« =x » indique que le mot doit se positionner exactement à x octets après la fin du précédent (ou début de phrase).

« > x» indique qu'il doit se positionner au moins à x octets après.

Par exemple, si le mot doit être absolument en tête de phrase (soit 0 octets après le début), on note `=0`. Si c'est n'importe où dans la phrase, on note `>0`.

Par exemple:

`>0 cat:chiffre`	Le premier morphe de la chaîne (un chiffre) peut être n'importe où dans la phrase
`=0 cat:specNum`	le deuxième (de la catégorie specNum) se place exactement 0 caractères après, autrement-dit immédiatement après
`=0` の	の doit être positionné immédiatement après le specNum
`>2 cat:vRad`	doit être positionné à au moins 2 caractères après の

`=0 cat:conjTa`	la conjugaison conjTa vient immédiatement après vRad.

Attention:
- Le chiffre désigne le nombre d'octets. Dans un encodage en shift-jis par exemple, un caractère japonais vaut deux octets.

G.2.3.2 Description des composants de la chaîne à chercher

G.2.3.2.1. Morphe unique

Si le composant est un simple morphe, l'indiquer là sans autre forme de procès (cf. の dans l'exemple précédent).

Note : lorsque deux morphes se suivent, il est inutile de les considérer séparément. au lieu de chercher :

```
>0 の
=0 何か
```

Il revient au même et il est plus rapide de chercher :

```
>0 の何か
```

G.2.3.2.2. Classe de morphes

Si plusieurs morphes peuvent occuper une position donnée dans la chaîne et que ces morphes sont lemmatisés, décrire la catégorie à l'aide d'une formule de traits comme suit :

```
cat:nom_de_la_catégorie
```

Il n'y a aucune contrainte d'écriture, contrairement au représentations syntaxiques qui dans le lexique sont strictement limitées aux conjonctions.

Cette liberté d'écriture permet de créer " à la volée " de nouvelles catégories sans avoir à modifier le lexique.

Exemple :

Etant donnés dans le lexique

les signes de ponctuation	(`ponctuation`)
les caractères ascii	(`caractere:ascii`)
les particules casuelles	(`particuleCas`)
les particules de conjonction	(`particuleConj`)
les noms communs désignant des fleurs	(`nomCommun & fleur`)
les noms communs désignant des humains	(`nomCommun & humain`)

on peut chercher une chaîne composés de trois composants qui n'existent pas en l'état dans le lexique mais que l'on peut construire "à la volée" en combinant les catégories existantes ci-dessus :

`>0 cat:ponctuation & -caractere:ascii`	n'importe où dans le segment une ponctuation qui n'est pas au format ASCII
`=0 cat: nomCommun & (humain | fleur)`	suivi immédiatement après d'un nom commun qui désigne soit un humain soit une fleur
`=0 cat:(particuleCas|particuleConj) & -が & -の`	suivi immédiatement après d'une particule casuelle ou de conjonction, à l'exclusion des particules が et の.

L'utilisateur trouvera d'autres exemples dans le manuel.

Le langage propositionnel de traits est décrit dans le chapitre I (p.40).

G.2.3.2.3. Sous-chaîne de contenu inconnu : `inconnu [longueur max=X]`

Il est possible d'extraire une sous-chaîne de contenu inconnu. Par exemple, on peut vouloir la sous-chaîne située entre deux morphèmes donnés. La sous-chaîne inconnue est signalée comme suit, sans indication sur la position :

```
inconnu [longueur max=X]
```

Typiquement, une requête comprenant une sous-chaîne inconnue se présentera sous la forme (`lemme`$_1$ et `lemme`$_2$ comportent comme montré plus haut les indications sur la position) :

```
<distance1> lemme1
inconnu[longueur max=X]
<distance2> lemme2
```

Si l'option `[longueur max=X]` est activée, le segment inconnu sera de longueur maximale *X*, même si la sous-chaîne inconnue est en réalité plus longue.

Dans tous les cas (option activée ou non), la longueur vaudra au plus 1024 octets.

Attention :

- Une chaîne inconnue ne peut être donnée comme unique composant d'un segment. Dans ce cas, SAGACE s'arrête.

- Un composant inconnu est nécessairement placé entre deux lemmes. Il ne peut débuter ou terminer une chaîne.

Améliorations envisageables :permettre à un inconnu de débuter ou terminer une chaîne.

Implémentation : lors de la recherche, le lemme inconnu n'est pas pris en compte (sauté dans work_s.c). Il est calculé lors de la sauvegarde.

G.2.3.2.4. Espaces dans la chaîne à chercher

L'espace (ascii: 20) est désigné par le mot réservé `espaceSimple`. Par exemple :

```
>0 EspaceSimple
```

G.2.3.3 Options d'affichage des résultats

Les options d'affichage présentées ici n'on pas d'effet pour l'affichage du concordancier au format KWIC. Elles n'ont d'effet que pour le concordancier non KWIC et pour les comptages.

Pour le concordancier KWIC, les options permettent spécifie ce qui sera affiché avec le segment trouvé. Par exemple, voici l'affichage en l'absence d'options :

Les options d'affichage ont des effets différents selon qu'il s'agit du concordancier

Les options d'affichage des résultats n'ont pas les mêmes effets selon qu'il s'agit d'une recherche d'exemples ou de collocations.

La gestion des options d'affichage a évolué et motivé le changement de version (de 3.x.x à 4.0.0). La principale modification est l'abandon de la conjugaison qui permettait d'afficher dans les résultats le lexème d'un radical conjugué du japonais (par exemple le lexème 食べる du radical 食べ, ou manger pour mangeait) au lieu du radical en question.

De même l'option d'affichage "lemme" disparaît et devient plus "rationnellement" un trait.

Plusieurs illustrations sont données plus bas dans cete section.

Attention : la notation des options tolérait quelques incartades. Désormais, une option d'affichage est obligatoirement précédée de /- alors que dans les versions antérieures, cela n'était imposé que pour la première option de la chaîne.

`/-affich:X` (concordancier non KWIC et collocation)

Les possibilités sont :

`affich:"X"` : affiche X.

`affich:trait:Y` c'est le trait du lemme qui s'unifie avec Y qui est affiché. Par exemple, si Y vaut "`traduction`" et que la description syntaxique du lemme comprend le trait valué "`traduction:"voiture"`" ", alors le trait "`traduction:"voiture"`" sera enregistré (et compté). Si deux lemmes différents on une même valeur de trait, les occurrences des deux lemmes seront additionnées. Par exemple 自動車 et 車 ont le même trait "`traduction:"voiture"` ". Si c'est ce trait qui est retenu, les occurrences des deux lemmes seront cumulées dans un résultat unique.

Lorsque le trait à afficher est absent de la représentation de l'entrée, la valeur affichée est `NO_VAL`

Les options `espaceDevant`, `affich:conjug`, `affich:lemme` et `affich:cat` ne sont plus valables après la version 4.0.0.

Voici des exemples de requêtes et de résultats :

Supposons que dans le lexique les entrées de la catégorie noCommun soient :

```
lemme  [[ cat & ref=REF & lexeme:LEXEME & LEMME & lemme:LEMME ]]
```

par exemple :

```
社会  [[ nomCommun & ref=1 & lexeme:社会 & 社会 & lemme:社会 ]]
飲み込 [[ radicalVerbal & ref=2 & lexeme:飲み込む & 飲み込 & lemme:飲み込  ]]
```

Voici les requêtes et leur effet :

Requête :	`>0 cat:nomCommun /-affich:trait:ref /-affich:trait:lexeme =0 の`
Résultat :	`::>1 社会`

Il est possible d'introduire des espaces, qui se positionneront aux endroits indiqués :

Requête :	`>0 cat:nomCommun /-affich:trait:ref /-affich:" " /-affich:trait:lexeme =0 の /-affich:" の"`
Résultat :	`::>1 社会 の`

Pour obtenir l'affichage de la forme conjuguée à la place d'un radical, il suffit d'afficher le lexeme (ou la valeur d'un autre trait dans lequel est enregistrée la forme :

Requête :	`>0 cat:radicalVerbal /-affich:trait:lexeme`
Résultat :	`::>飲み込む`

`/-compte` (recherche de collocations)

Le comptage peut être activé pour tout ou partie des morphes d'une chaîne cherchée. Pour que cette commande soit viable, il est impératif que la commande affich:X soit aussi activée. En l'absence de cette dernière, SAGACE s'arrête et signale l'absence. Voici une explication progressive de la façon dont il faut interpréter les chiffres dans l'affichage des résultats.

Exemple 1:

Soit la requête et le résultat suivants (les chiffres indiqués sont fantaisistes ; seul le fait qu'ils soient décroissants est réaliste).

requête	résultat
>0 cat:chiffre /-affich:trait:lemme /-compte =0 cat:specNum /-affich:trait:lemme /-compte =0 の >2 cat:vRad /-affich:trait:lemme /-compte =0 cat:conjTa /-affich:trait:lemme /-compte	... 何 10 本 8 まと 3 めた 1 一 2000 層 400 求 30 めた 3 ...

Détaillons la première ligne de résultats ci-dessus.

Sur N occurrences de la chaîne,

- 何 apparaît 10 fois
- 本 apparaît 8 fois derrière le chiffre 何
- まと apparaît 3 fois derrière la chaîne 何本
- めた apparaît 1 fois derrière la chaîne 何本まと

Exemple 2 :

Pour étudier l'occurrence d'un morphe A indépendamment de la valeur que prend un morphe B qui le précède, il suffit de ne pas afficher B dans le résultat. On comprendra alors que l'on compte B en tenant compte de la présence de la catégorie des morphes B, mais sans tenir compte de la valeur exacte de ces morphes. Par exemple :

>0 cat:chiffre =0 cat:specNum =0 の >2 cat:vRad /-affich:trait:lemme /-compte =0 cat:conjTa /-affich:trait:lemme /-compte	... まと 200 めた 10 求 40 めた 30 ...

Détaillons la première ligne de résultats :

- まと apparaît 200 fois derrrière une chaîne (discontinue) commençant par un chiffre (quel que soit celui-ci) suivi d'un specNum (quel que soit celui-ci) suivi du morphème の.

- めた apparaît 10 fois après la chaîne <chiffre+specNum+の+...+まと>. On peut seulement déduire qu'il apparaît au moins autant de fois dans le corpus, derrière まと mais on ne peut pas dire combien de fois exactement, en tout.

`/-compteRef`(recherche de collocations)

Les collocations listées seront ordonnées en fonction du nombre d'occurrences du lemme marqué par cette option.

Implémentation : Dans un premier temps, le résultat est enregistré dans le fichier FTEXTE donné dans la requête. A la fin de la recherche, renomme FTEXTE en FTEXTEtmp, sauve les données ordonnées dans FTEXTE et efface FTEXTEtmp.

`/-KWIC1` **(concordancier au format KWIC)**

Pour l'enregistrement au format KWIC, c'est le premier composant de la chaîne cherché qui par défaut est centré. Pour prendre comme centre un autre composant, insérer dans la chaîne d'options d'affichage l'option `/-KWIC1`.

Voici par exemple une requête :

>0 cat:particule | ponctuation =0 cat:motAbsent /-affich:trait:lemme =0 cat:particule|ponctuation				>0 cat:particule | ponctuation =0 cat:motAbsent /-affich:trait:lemme/-KWIC1 =0 cat:particule|ponctuation			
||心なき林	/	の	/ 木木も||相	、たしかに	/	数数	/ の不思議が
その奉仕	/	の	/ 品品が、入	、枕もとの	/	品品	/ を見ました
くと、奉仕	/	の	/ 品品の饗應	すべてが、	/	様様	/ の着物を着
は、枕もと	/	の	/ 品品を見ま	その奉仕の	/	品品	/ が、入口の
も、たしか	/	に	/ 数数の不思	だして緑の	/	木木	/ の間には色
も花であり	/	、	/ 樹樹も花で	と、奉仕の	/	品品	/ の饗應にあ
やらの、そ	/	の	/ 樹樹のまわ	らの、その	/	樹樹	/ のまわりを
れだして緑	/	の	/ 木木の間に	花であり、	/	樹樹	/ も花であり

人すべてが / 、 / 様様の着物	心なき林の / 木木 / も||相凭り

Dans le cas de gauche, en l'absence de précision, c'est par défaut le premier composant qui est centré.

(dans cet exemple l'ordre des réponses n'est pas le même car les phrases sont ordonnées "alphabétiquement").

`/-ordreAffichage:X`

0<X

Par défaut, les composants à afficher dans le résultat sont affiché dans leur ordre d'apparition. Cette option permet de changer l'ordre.

Par exemple :

`Reqet:`
`>0 x /-affich:»x»`
`=0 y /-affich:»y»`
	`Si le patron est trouvé,`
	`le résultat sera : xy`

`Reqet:`
`>0 x /-affich:»x» /-ordreAffichage:2`
`=0 y /-affich:»y» /-ordreAffichage:1`
	`Si le patron est trouvé,`
	`le résultat sera : yx`

G.2.4. Recherche multiple

A partir de la version 4.???.0, il est possible de lancer une recherche sur plusieurs stuctures. Par exemple, chercher les noms communs précédés d'un groupe numéral et précédés d'un nom. Pour cela, il suffit de multiplier autant de fois qe nécessaire les structures recherchées et à chaque fois d'indiqer de répéter l'entête « Reqet: ».

Par exemple :

`<<<ENTETE DE REQUETE>>>`	

`Reqet:`
`>0 cat:nomCommun`
`=0 の`
`=0 cat:nomCommun /-affich:lemme/-compte`
	`Comptage des noms communs précédés d'un autre nom commun`

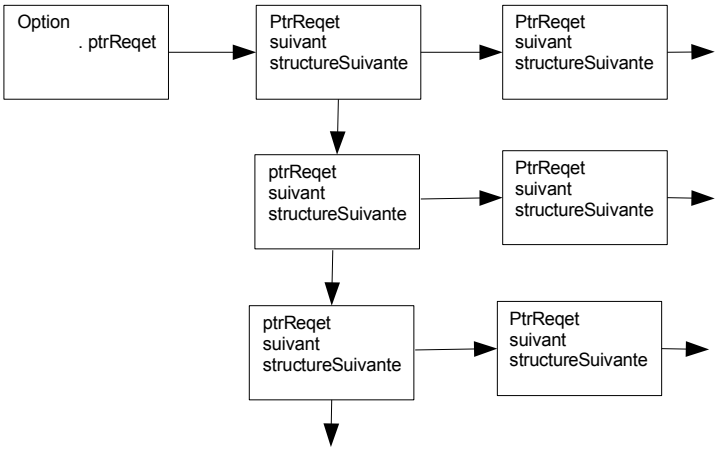
`Reqet:`
`>0 cat:specNum`
`=0 の`
`=0 cat:nomCommun /-affich:lemme/-compte`
	`Comptage des noms communs précédés d'un spécifique numéral`

Les résultats sont combinés dans une même liste.

Historique : initialement, les recherches sur des structures différentes étaient lancées séparément et les résultats combinés par des routines externes. Mais il s'est avéré que ce procédé était coûteux. Il faut en effet accéder aux fichiers du corpus et les lire autant de fois qu'il y a de structures à étudier, d'où une perte de temps non négligeable certainement. Puis il faut créer et gérer des routines externes, ce qui est un surcroît de tâches à implémenter, et autant de risques d'erreurs.

G.2.5. Implémentation des chaînes cherchées

Chaque patron est décrit à l'aide d'une chaîne de ptrReqet (horizontal). A chaque composant correspond un ptrReqet.


Option
. ptrReqet
PtrReqet
suivant
structureSuivante
PtrReqet
suivant
structureSuivante
ptrReqet
suivant
structureSuivante
PtrReqet
suivant
structureSuivante
ptrReqet
suivant
structureSuivante
PtrReqet
suivant
structureSuivante

H. Paramétrage de la requête pour un texte structuré

Pour simplifier, au lieu de "recherche qui tient compte de la structure du texte" on dira *recherche structurée*. La présentation du fichier de requête ne diffère que sur quelques détails. Voici typiquement sa forme :

Structure	Exemple
En-tête : Passage des paramètres de la requête	`<l'entête est élaboré selon le même principe que pour une recherche dans un texte non structuré>`
Chaîne à chercher dans un segment de rang donné (autre que le corps du texte)	`<TITRE...>` `description de la sous-chaîne`
Chaîne à chercher dans tout segment de rang inférieur	`Reqet:` `description de la sous-chaîne`

Il est possible avec SAGACE de chercher une sous-chaîne à deux composants distants, l'un situé dans un titre, l'autre situé dans tout titre de niveau inférieur à celui du premier composant ou dans le corps du texte. Par exemple, pour chercher tous les adjectifs qui sont dans la portée d'un titre comprenant un mot donné : chercher tous les adjectifs associé au nom d'une entreprise par exemple.

Ce type de recherche n'est valable que pour les textes balisés avec informations sur la structure du texte et titrage (voir chap.E.3.2)

Au niveau du corps de texte, les segments sont ceux délimités par le symbole d'arrêt donné dans l'entête. L'utilisateur se reportera donc au chapitre précédent pour plus d'informations.

Nous nous concentrons ici sur la description du titre dans la requête.

Avec une marque de la forme `<TITRE=`*x*`>` et *x* compris entre 1 et 9, seul les titres de rang *x* sont traités. Les autres titres ne sont pas pris en compte. La recherche porte ensuite sur le corps du texte.

Avec une marque de la forme `<TITRE=<`*x*`>` et x compris entre 1 et 9, seuls les titres de rang 1 à *x* sont pris en compte. Les titres de niveau inférieurs (de valeur supérieure) ne seront pas lus.

Avec une marque de la forme `<TITRE=>`*x*`>` et *x* compris entre 1 et 9, seuls les titres de rang *x* à 10 sont pris en compte. Les autres titres ne sont pas pas lus.

Implémentation : dans l'imprémentation des options générales, l'entête est lu jusqu'à détecter soit "Reqet" soit "<TITRE". Si c'est le premier cas, alors on déduit qe c'est une recherche dans un texte non structuré. Dans le deuxième cas, c'est une recherche dans un texte structuré.

Si c'est une recherche structurée, relève l'interval exploitable.

I. Langage propositionnel de traits, LEXSPROP

Les représentations syntaxiques sont décrites à l'aide de formules de la logique des propositions. Celles-ci permettent une manipulation très intuitive et aisée des catégories. Nous présentons en deux temps ce langage, baptisé LEXSPROP : sa syntaxe, puis sa sémantique. Nous évoquons ensuite le mode de calcul.

I.1. Syntaxe

I.1.1. Définition

Il s'agit d'un langage des propositions. La seule « originalité » tient à la rédaction des constantes propositionnelles, beaucoup plus libres que ce qui se pratique traditionnellement.

Constantes propositionnelles (ou *traits*) :

(a) chaîne de caractères quelconques entre deux guillemets " " (unicode U0022) et ne contenant pas de guillemets

(b) n'importe quelle chaîne de lettres ou de caractères latins non accentués

(c) combinaison de (a) ou (b) avec les chiffres ou les caractères ':' , tiret, barre basse : '_'

(d) n'importe quelle combinaison de (a), (b) et (c)

Connecteurs

négation (-) , conjonction (&) et disjonction (|)

Formule propositionnelle bien formée de traits (T et ◊ sont respectivement n'importe quel trait et connecteur, P est une formule bien formée) :

1) P ← T

2) P ← -P

3) P ← P ◊ P

4) P ← (P ◊ P)

En l'absence de parenthésage, une négation porte sur la constante qu'elle précède :

(-a | b) vaut ((-a) | b) et non (- (a | b))

Ici, l'élimination des parenthèses est permise syntaxiquement pour simplfier, mais en fait elle se déduit de la sémantique de la logique des proposition.

Le langage décrit par cette grammaire sera dit " langage propositionnelle de traits ".

I.1.2. Exemple de traits propositionnels

a abcde forme:genre:fem «je suis la traduction du lemme»

N'étant pas décomposés, les traits avec ":" sont pour l'instant sous-exploités. Le but est à terme de manipuler des couples de traits:valeur. La quatrième forme de trait permet d'introduire dans le lexique des données quelconques, sans limites de formes. Elle est aussi utile pour manipuler des caractères non ascii qui pourraient être mal interprétés par le parseur.

I.1.3. Exemple de formules

Voici des exemples pour illustrer les formules qu'autorisent les règles de la grammaire.

1) N'importe quel trait est une formule de traits.

2) La négation d'une constante ou d'une proposition :

-nom ou -(nomPropre & -(forme:genre:fem | forme:masc:fem))

Une négation peut bien sûr entrer dans la portée d'une autre négation.

3,4) Les deux règles permettent en fait de s'affranchir de la rédaction des parenthèses, lorsque cela n'entraîne pas d'ambiguité. Par exemple :

(c & c) peut être rédigé sans parenhèses : c & c

C'est particulièrement intéressant pour des formules ne comportant qu'un type de connecteur :

c_1 & c_2 & -c_3 ... vaut par associativité aussi bien ((c_1 & c_2) & -c_3) que (c_1 & (c_2 & -c_3))

Voici un autre exemple où les parenhèses sont réduites au strict minimum, de sorte à désambiguïser la formule :

(a_1 & a_2 & a_3) | (b_1 & b_2 & b_3) | ...

I.2. Interprétation informelle des formules propositionnnelles de traits

Un trait se comprend comme un nom d'ensemble. A un trait ne correspond qu'un seul ensemble.

On se donne un ensemble $\mathcal{D}$ l'union de tous les ensembles désignés par un trait du lexique.

Une conjonction de la forme (A & B) se comprend comme l'intersection des ensembles désignés par A et B.

Une disjonction de la forme (A | B) se comprend comme l'union des ensembles désignés par A et B.

Une négation de la forme -A se comprend comme l'ensemble complémentaire de A dans $\mathcal{D}$.

Un lemme peut être vu comme l'élément de l'ensemble décrit par la conjonction de traits qui lui est associée. Une catégorie est un ensemble de lemmes.

Ainsi, noter dans le lexique

```
私  [[ nom & prenom & traduction:francais:val:"je" ]]
```

s'interprète

> le lemme 私 appartient à l'intersection des trois catégories désignées par les traits nom, prenom et traduction:francais:val:"je"

Conformément à la sémantique de la logique des propositions, l'interprétation est une fonction. Soit σ une fonction d'interprétation telle que

```
σ : traits -> catégories
```

on reformule la relation d'appartenance comme suit :

```
私 ∈ σ(nom) ∩ σ(prenom) ∩ σ(traduction:francais:val:"je")
```

Comme indiqué (chap.Erreur : source de la référence non trouvée, p.F.4), chaque lemme devient trait de sa propre catégorie puisque pour tout lemme L de catégorie C, SAGACE ajoute automatiquement le trait L à la représentation syntaxique de L. Un lemme L appartient donc au moins à l'ensemble désigné par lui même.

Une conséquence est qu'une catégorie désignée par la formule de trait - の désigne l'ensemble complémentaire de l'ensemble désigné par σ(の). Il s'agit en fait du lexique entier moins les entrées qui ont le trait-lemme の.

Pour d'autres exemples d'interprétation, voir le chap.L.3 p.49.

I.3. Calculs sur les formules de traits

I.3.1. Calcul de base

Lorsque d'un composant de chaîne est décrit sous la forme d'une catégorie, SAGACE liste avant la recherche tous les lemmes du lexique qui appartiennent à cette catégorie. Lors de la recherche, il cherchera une occurrence de l'un de ces lemme. De la sorte, l'étude de l'appartenance consiste ni plus ni moins à vérifier que

```
(1) catégorie_du_lemme ⊆ catégorie_désirée
```

Concrètement, soient les formules Freqet et Flemme décrivant respectivement la catégorie désirée (indiquée dans la requête) et la catégorie du lemme consulté (dans le lexique), le calcul a lieu comme suit :

1) Mise sous forme normale disjonctive de Freqet (désignons la formule ainsi obtenue par fnd(Freqet)). Du point de vue de l'interprétation ensembliste, cela revient à décrire la catégorie_désirée comme une union

d'intersections. Mettons que fnd(Freqet) se présente sous la forme : $C_1 \mid C_2 \mid ... \mid C_n$ avec C_i ($i \in [1,n]$) des conjonctions de traits.

2) En théorie des ensembles, la relation d'inclusion (1) est vérifiée si il existe au moins une conjonction C_i telle que $\sigma(Flemme) \subseteq \sigma(Ci)$. $\sigma(Flemme)$ étant une intersection, il suffit de vérifier que $\sigma(Flemme)$ est l'intersection d'au moins tous les ensembles dont $\sigma(Ci)$ est elle-même l'intersection. D'un point de vue pratique, il suffit pour cela de vérifier que chaque trait non nié dans Ci est présent dans Flemme et de vérifier que chaque trait nié dans Ci est absent de Flemme.

Exemple :

Soit Flemme=(nomCommun & humain & ひと) et soit Freqet=nomCommun. On cherche à savoir si $\sigma(Flemme) \subseteq \sigma(Freqet)$.

1) Mise sous forme normale disjonctive de Freqet, la formule de la requête ne change pas : fnd(Freqet)=nomCommun.

2) L'inclusion est évidente mais on peut la vérifier par manipulation des traits : Freqet contient une unique conjonction, constituée d'un unique trait. Cet unique trait est présent dans Flemme. On a bien une inclusion. Le lemme appartient à la catégorie $\sigma(Freqet)$.

Exemple avec négation

Soient Flemme identique à l'exercice précédent, et Freqet = -humain.

1) fnd(Freqet) = -humain

2) En termes ensemblistes, on doit vérifier que $\sigma(Flemme)$ appartient au complément de $\sigma(humain)$, c'est à dire qu'il ne doit avoir aucune partie commune avec $\sigma(humain)$. Cela est vérifié si le trait humain n'apparaît pas dans Flemme.

Exemple

Soient Freqet=nomCommun & -humain . Il s'agit intuitivement de l'ensemble des noms communs qui ne sont pas humains.

1) fnd(Freqet) = Freqet

2) Testons les traits de l'unique conjonction. nomCommun est bien présent mais humain l'est aussi. L'inclusion n'est donc pas vraie.

Exemple

Soient Freqet = nomCommun & -anime . Il s'agit intuitivement de l'ensemble des noms communs qui ne sont pas animés.

1) fnd(Freqet) = Freqet

2) Testons les traits de l'unique conjonction. nomCommun est bien présent dans Flemme mais pas anime. L'inclusion n'est donc pas vérifiée.

I.3.2. Calcul par défaut

Le calcul effectué est implicitement un calcul par défaut. En effet, lorsque nous étudions la relation :

$$nomCommun \subseteq nomCommun \cap humain$$

nous aboutissons par le calcul évoqué ci-dessus à une réponse négative. Cela suppose par défaut que humain $\neq$nom. Ce n'est dit nulle part explicitement mais c'est en définitive une donnée implicite.

I.3.3. Simulation par la logique des propositions et la négation par échec

La syntaxe du langage et le calcul avec recours à des valeurs par défaut permet de simuler les calculs ensemblistes ci-dessus par le calcul logique propositionnel avec négation par échec.

A ce stade de développement de SAGACE, nous n'exploitons pas à fond les techniques de calcul que ce constat nous permettrait de faire.

J. Implémentation

L'implémentation n'est pas optimisée

Le lexique est implémenté sous la forme d'un arbre digitale dont chaque feuille correspond à un octet. Cette stratégie était initialement motivée par le besoin de s'affranchir des encodages des textes manipulés et du lexique.

Traitement de la requête

A partir de la version 3.4.0, la requête est enlistée (longueur maximum de 100 lignes). Le listage est effectué dans interpreteReqet.c .

Implémentation de l'arbre digital

Il s'agit d'un arbre digital non compressé. Pour être indépendant de l'encodage, c'est chaque octet qui est porté par une feuille et non pas les caractères, puisque ceux-ci sont encodés diffféremment selon l'encodage.

Les feuilles de l'arbre sont des structures de la forme :

```
char *val
struct chaineTraits *chaineTrait
```

(la structure chaîneTraits est ici donnée de façon simplifiée ; il s'agit en fait d'une chaîne de pointeurs PTR_CHAINE2_CHAR_AS)

Un lemme (a_1 ... a_n) comportant une description syntaxique construite avec une conjonction de traits (t_1:v_1 & ... & t_m:v_m), et si dans la requête il est indiqué que le trait ti:vi doit être comptabilisé, alors le lemme sera implémenté comme suit :

0	-	a1	-	...	-	an	-	NULL
NUL		NULL		...		NULL		$trait_i$:val_i

(Les traits listé sont ceux indiqués dans la requête)

K. Ligne de commande

SAGACE ne peut être lancé qu'en ligne de commande et ne dispose pas d'interface graphique.

`./sagace -h`
Aide en ligne

Important : Pour toute action suivantes, SAGACE a besoin de consulter le fichier de requête. En conséquence, toute commande est de la forme :

`./sagace ... [ -f cheminEtNomDeFichier] ...`

Si l'option `-f cheminEtnomDeFichier` n'est pas présente, SAGACE exploite par défaut le fichier `requet.txt` présent dans le répertoire que l'exécutable `sagace`. Sinon, il exploite le fichier indiqué par `cheminEtnomDeFichier`.

Dans les descriptions qui suivent, le " lexique " désigne celui indiqué dans le fichier de requête.

`./sagace [-v][-f cheminEtnomDeFichier]`
Lance la requête. Si le mode verbose est activé (désactivé par défaut) indique le nom du fichier en cours de lecture. Pour une recherche de type concordancier, affiche un point à chaque fois qu'un exemple est trouvé.

`./sagace {-a|--nbEntrees} [-f cheminEtNomDeFichier]`
Donne le nombre total d'entrées du lexique. Le comptage est assez grossier puisqu'il se contente de compter toutes les lignes des fichiers du lexique qui ne sont pas nulles (après élimination des commentaires) et qui ne sont pas une donnée de catégorie (lignes commençant par "`cat:`").

`./sagace -{b | d | e} [-v] [-f cheminEtNomDeFichier]`

`-b` `--eg`	Liste les catégories du morphe indiqués par la première ligne de la requête.
`-d` `--deb`	Liste les mots qui commencent par le mot indiqué par la première ligne de la requête et donne leur catégorie.
`-e` `--in`	Liste les mots qui comprennent par le mot indiqué par la première ligne de la requête et donne leur catégorie.
`-g` `-fin`	Liste les mots qui finissent par le mot indiqué par la première ligne de la requête et donne leur catégorie.
`-v`	Indique le fichier où est enregistrée l'entrée trouvée.

`./sagace [-f cheminNomFIchierReqet] --inserBasicLexic "entree"`
Insère l' "entrée" en fin du fichier `ajout.dic` situé au niveau de la racine du répertoire du lexique. Le fichier ajout.dic est créé si il n'existe pas.

`./sagace {-m|--listMot} [-n x][-f cheminEtNomDeFichier]`
Liste toutes les lemmes du lexique qui appartiennent à la catégorie indiquée dans la requête. Limite éventuellement aux x premières entrées.

`./sagace {-t,--listeTraits} [-f cheminEtNomDeFichier]`
Liste les traits du lexique.
Les traits ne sont pas relevés si ils commencent (ou se réduisent à) :

commentaire:	traduction
ecriture:val	semantic:marqueCommercialeDe:
lecture:val	source:

divers:	abreviation
syntaxe:hyperonyme	traduction
date:	specNumCompatible
commentaire:	semantique
auteur:	convertible
syntaxe:hyperonyme	

C'est une liste *ad hoc*. Actuellement, elle apparaît dans le code pour des raisons de simplicité mais il faudrait proprement les lister dans un fichier à part pour être modifiable à la demande.

```
./sagace {-cc|--listCatComplet} [cheminEtNomDeFichier]
./sagace {-c|--listCat} [cheminEtNomDeFichier]
```

Ces commandes ne sont plus disponibles à partir de la version 3.0.0 .

Améliorations envisageables : ne pas implémenter la liste en dure et la rendre accessible à l'utilisateur.

L. Tutoriel (SAGACE-v4.0.0)

Ceci est un tutoriel simple de prise en main du logiciel SAGACE, version individuelle. Il prend exemple sur le japonais mais est facilement compréhensible pour des non japonisants.

Il est basé sur la version 4.0.0 .

L.1. Installation

On suppose que le logiciel a été installé correctement et qu'il se trouve dans le répertoire REP. L'exécutable et le fichier de requête par défaut sont dans ce répertoire.

Nous utilisons le corpus aozora et le lexique LEXS-J-β3.X.X , tous les deux récupérés sur l'internet.

Le fichier `LEXS-J-b3.X.X.tar.gz` est décompressé dans le répertoire `REP/dico`. Le lexique se trouve alors dans `REP/dico/donnees`.

Le corpus Aozora est installé dans le répertoire `REP/corpus`.

On construit le descripteur de corpus nommé `scriptCorpus.txt` :

```
groupe: aozora
REP/corpus/
```

Le descripteur est enregistré dans le répertoire courant REP.

Pour toutes les recherches, on utilisera le fichier `reqet.txt` par défaut. Dans ce fichier, on peut dors et déjà renseigner quelques lignes parmi celles qui sont obligatoires :

```
RepertoirLexic:REP/dico/donnees/
ScriptCorpus:REP/corpus/corpus.txt
NomDuGroupe:aozora
```

Puisque nous travaillons seulement sur le japonais et que nous prenons comme unité de texte la phrase (qui se termine par un rond "。", nous pouvons aussi compléter les lignes :

```
Arret:"。"
Langue: japonais
```

Paramétrons les options non obligatoires. Pour ce tutoriel, nous nous contenterons à chaque fois d'une dizaine d'exemples, sans se donner la peine d'interdire le travail sur des occurences multiples de phrases. Par contre nous voulons savoir à partir de quels fichiers sont extraits les exemples (dans le cas de recherche d'exemples) :

```
Affiche le nom du fichier d'origine
Nombre max des exemples:10
```

L.2. Travail sur des occurrences de chaînes d'un seul lemme

L.2.1. Recherches de segment contenant un seul lemme donné

Il faut indiquer le mode de recherche suivante : `exemple`.

Chercher dix phrases contenant

1- le mot 私 en tête de phrase.

Dans le fichier de requête, la chaîne est décrite sur une simple ligne se présentant :

```
=0 私
```

On comprend cette ligne : SAGACE recherche une chaîne d'un seul lemme ; ce lemme est situé à exactement (=) 0 octets après le début de la ligne. En d'autres termes, il est en tête de segment (lequel est une phrase finissant par un "。").

Voici le fichier complet :

```
Travail: exemple
Resu:REP/resu.txt
RepertoirLexic:REP/dico/donnees/
ScriptCorpus:REP/corpus/corpus.txt
NomDuGroupe:aozora
Arret:"。"
Langue: japonais
Affiche le nom du fichier d'origine
Taille max des exemples:10
Reqet:

=0 私
```

En l'état, chaque ligne trouvée sera affichée :

```
::>
fichier_source
phraseEtudie : 私は....
```

Etant donné que le lemme cherché est connu d'avance, il n'y a pas grand intérêt à l'afficher, mais rien n'empêche de le faire. La commande :

```
=0 私 /-affich:trait:lemme
```

produira :

```
::> 私
fichier_source
phraseEtudie : 私は....
```

2- le mot 私 n'importe où dans la phrase.

```
>0 私
```

Pour obtenir le morphème à au moins plus d'un caractère de distance du début de phrase, sachant qu'un caractère en SJIS est codé sur 2 octets :

```
>2 私
```

3- n'importe quel pronom en tête de phrase.

La seule différence d'avec (1) est qu'on ne travaille pas sur un lemme unique mais sur un ensemble/catégorie de lemmes. Il faut donc indiquer que c'est une catégorie (`cat:`) qui est cherchée, et donner les traits qui caractérisent cette catégorie. Pour connaître les traits, il suffit de consulter la liste grâce à la commande suivante :

```
./sagage -t
```

qui affiche la liste des traits utilisés dans le lexique. Cette commande suppose que le fichier de requête a été renseigné correctement, en particulier la localisation du lexique. Ceux de LEXS-J-β3.X.X ayant des noms relativement intuitifs, on repère sans problème "pronom". En cas de doute, on listera le contenu de la catégorie désignée par ce trait.

Remplissons la requête :

```
=0 cat:pronom /-affich:trait:lemme
```

On veut à 0 octets après le début de la phrase un lemme dont la catégorie est décrite par la formule (très simple) à un seul trait "pronom". Dans le résultat, on veut en plus afficher le pronom trouvé pour cette catégorie.

Le fichier requête étant rempli, listons le contenu de la catégorie pronom. Pour cela, on utilise la commande :

```
./sagace -m
```

SAGACE liste alors la totalité des lemmes qui appartiennent à la catégorie désignée sur la première ligne de la chaîne de requête dans la chaîne de requêtes. On se doute que le nombre de lemmes est faible. Dans le cas d'une catégorie volumineuse, il peut être intéressant de limiter l'affichage aux 20 premiers lemmes (ou toute autre limite). Il suffit d'ajouter :

```
./sagace -m -n 20
```

Puisque les lemmes ainsi obtenus correspondent à ce qu'on attend, il n'y a plus qu'à lancer la requête.

4- n'importe quel pronom n'importe où dans la phrase.

Sans surprise :

```
>0 cat:pronom /-affich:trait:lemme
```

On aura vraiment ici intérêt à afficher le lemme trouvé sans quoi la lecture du résultat sera laborieuse, en particulier dans une phrase longue lorsqu'il faut deviner quel pronom a été trouvé et où il est.

L'affichage reste cependant assez rustique. L'idéal sera à terme de marquer le lemme trouvé, directement dans le segment.

Refaire la même chose mais en limitant la longueur des phrases à 50 caractères japonais (choisir "longueur 100").

Ces quelques essais font apparaître les limites de SAGACE. En l'absence d'analyse syntaxique globale du segment, l'occurrence trouvée peut ne pas correspondre au lemme cherché. Reprenons l'exemple (4) et un des segments obtenus :

```
::> われ
doreikonjo_ron1.txt:

phraseEtudiee: そして捕虜は駄獣として農業の苦役に使われた。
```

Dans cette phrase, l'occurence de **われ** (soulignée par nous) n'est pas un pronom. Elle n'a même aucun statut morphologique. Pour interdire ce type d'erreur, il est recommandé de contraindre l'environnement du lemme. Nous verrons ça plus bas (voir chap.L.5, p.50).

L.2.2. Listage des occurrences d'un lemme donné

Il faut indiquer le mode de recherche suivante : stat.

1- Comptage du mot 私 en tête de phrase (c.-à-d. combien de fois le mot apparaît en début de phrase dans la collection de textes de Aozora).

```
=0 私 /-compte
```

2- Comptage du nombre de phrases dans le corpus

Une phrase est délimitée par le rond et il n'y en a qu'un par phrase (étant donné le mode de définition adopté pour les segments de travail). On obtiendra donc indirectement le nombre de phrases en connaissant le nombre de ronds dans le texte.

```
>0 。 /-compte
```

La version exploitée en contient 124 020 .

3- Comptage du mot 私 n'importe où dans la phrase (c.-à-d. combien de fois le mot apparaît dans le corpus).

```
>0 私 /-compte
```

4- Comptage de toutes les formes de pronoms en tête de phrase

```
=0 cat:pronom /-compte
```

Ce n'est pas possible d'avoir un résultat directement. Il faut additionner les occurences de chaque lemme trouvé pour la catégorie. Une autre solution est donnée plus loin dans le tutoriel. (voir chap.L.4, p.49).

5- Refaire la même chose mais en limitant la longueur des phrases à 50 caractères japonais (choisir "longueur 100")

L.3. Manipulation du langage propositionnel de traits

Pour voir le résultat des manipulations sur le langage, il suffit d'écrire la formule dans le fichier de requête et d'afficher le contenu de la catégorie avec la commande "-m".

1) Décrire la catégorie des pronoms désignant des humains (vs pronom décrivant un lieu, une date etc.)

On comprend que c'est nécessairement une sous-catégorie des pronoms. Comme on ne connaît pas l'ensemble des traits utilisés pour décrire cette catégorie, le plus simple est (a) de choisir un pronom qui désigne un humain, (b) voir la description de sa catégorie et relever le(s) trait(s) qui de toute évidence concerne(nt) son "humanité", (c) éventuellement vérifier que ce trait correspond à ce que l'on attend en listant les mots qui ont le trait "humain" (c.-à-d. les mots de la catégorie décrite par le seul trait humain), puis (d) construire la formule. Voici le détail de chaque étape :

(a) affichage de la catégorie du mot 私 qui est un pronom désignant un humain.

dans la requête : `=0 私`

commande : `./sagace -b`

(b) Il y a trois réponses, qui ne se distinguent que par la lecture. Quant à la catégorie, on y trouve le trait "pronom" et "humain".

(c) Par acquis de conscience, vérifions que le trait "humain" ne correspond qu'à des humains :

dans la requête : `=0 cat:humain`

commande : `./sagace -m`

C'est le cas. On constate d'ailleurs qu'il n'y a que des pronoms.

(d) La catégorie est l'intersection de celle des pronoms et de celle des lemmes désignant des humains (trait "humain"). On obtient la catégorie décrite par la formule : pronom & humain.

2) Donner la représentation des noms qui sont des noms communs.

Sachant que dans LEXS-J-β3.X.X, les noms communs sont un sous ensemble de noms, on peut se contenter du trait : nomCommun. Si pour une raison quelconque on fait du zèle : nom & nomCommun

3) Donner la représentation des lemmes qui ne sont pas des noms communs.

```
-nomCommun
```

4) Représentation des noms qui ne sont pas des noms communs.

On décompose la description de la catégorie : c'est un lemme qui est un nom (`trait : nom`) et (&) c'est un lemme qui n'est pas un nom commun (`-nomCommun`) : `nom & -nomCommun .`

5) Représentation de l'ensemble des particules autres que la particule の (sachant que parmi les traits de tout lemme figure le lemme lui-même).

```
particule & -の
```

6) Représentation du de l'ensemble des catégories autres que les particules の et が :

```
(particule & -が & -の)
```

Le langage de traits est un langage propositionnel avec ce que cela impliqe d'équivalence entre les formules. Ainsi, la formule ci-dessus est strictement équivalente à (particule & -(が|の)) .

L'équivalence entre les formules relève des connaissances de base de la logique formelle et elles s'acquièrent en peu de minutes. Elle est en général donnée dans n'importe quel manuel de logique. Sans aller jusque-là, cette logique étant basique et facile à interpréter, il suffit de s'en remettre à l'intuition et de procéder à quelques essais.

L.4. Recherche de chaînes composées

1) Recherche d'une chaîne de deux lemmes, 私 et は, n'importe où dans la phrase.

Il n'y a aucun intérêt dans le cadre d'une recherche avec SAGACE de séparer les deux lemmes d'une chaîne continue. En conséquence :

```
>0 私は
```

vaut

```
>0 私
=0 は
```

La seule différence est que la seconde recherche consomme plus de ressources (ce qui ne doit guère se ressentir dans les faits).

Dans les exemples suivants, nous étudions les combinaisons "spécifique numérique" + nom, qui sont des combinaisons que l'on trouve aussi bien en japonais qu'en chinois et coréen.

2) Chercher des phrases contenant les chaînes suivantes :

" tout chiffre à l'exception des chiffres inscrits dans un rond " + " spécifique numérique "

n'importe où dans la phrase. (chiffres : **chiffre** ; chiffre inscrit dans un rond : **chiffreDansRond** ; spécifique numérique : **specNum**).

```
>0 cat:chiffre & -chiffreDansRond
=0 cat:specNum
```

3) Idem : tout couple de chiffres suivi d'un spécifique numérique

```
>0 cat:chiffre & -chiffreDansRond
=0 cat:chiffre & -chiffreDansRond
=0 cat:specNum
```

Il va sans dire que l'absence d'une analyse morpho-syntaxique qui permettrait de générer un numéral de longueur qelconque est un sérieux handicap. En effet, pour étudier les chaînes avec des numéraux d'une longueur linguistiquement plausible (entre 1 et 5 vraissemblablement), il faut faire cinq recherches.

4) Idem : tout groupe numéral avec au moins un chiffre et un spécifique numérique en position déterminante devant un nom commun (chiffre + specNum + の + nom commun). N'afficher que le spécifique numérique et le nom commun. Dans le fichier de résultats, pour repérer immédiatement le couple trouvé dans l'exemple, on affichera uniquement le spécifique numérique et le nom commun. Sans autre précision, les deux seront présentés collés l'un à l'autre. Demander d'insérer un espace devant le nom commun.

```
>0 cat:chiffre
=0 cat:specNum /-affich:trait:lemme
=0 の
=0 nomCommun /-affich:” “ /-affich:trait:lemme
```

On exige ici que le groupe numéral se termine au moins par un chiffre. Celui-ci peut alors être précédé d'un ou plusieurs chiffres.

Comme on le voit dans les résultats, l'occurrence du lemme classé comme nom commun n'est dans certains exemples qu'une partie d'un nom.

5- Compter le nombre de phrases débutant par un pronom.

```
=0 cat:pronom
>0 。 /-compte
```

Le résultat est : 14847.

L.5. Contrôler l'environnement immédiat des lemmes dans le texte

En l'absence d'une analyse syntaxique générale de la phrase, le statut morpho-syntaxique d'une occurrence de lemme peut être sans rapport avec le lemme cherché. Nous l'avons vu pour les pronoms mais ce n'est bien sûr pas le seul cas. Par exemple, l'occurrence de 本 dans 日本語 n'a aucun rapport avec le nom commun 本. Une analyse morpho syntaxique aurait en principe interdit de décomposer 日本. Pour limiter les erreurs, une technique consiste à imposer des environnements qui "délimitent" le mot.

1) Désambiguïsation en contrôlant la distribution après le lemme.

Comptons les occurrences des pronoms en tête de phrase, en imposant la présence d'une particule derrière :

```
=0 cat:pronom
=0 cat:particule
```

```
>0 。
```

Le résultat obtenu est 12790. La position de la particule peut être occupée par des lemmes qui n'ont en fait pas le fonctionnement de particule à cet endroit. Rien n'empêche de préciser la nature des particules.

2) Lorsque le lemme est en milieu de phrase, on peut lui imposer d'être précédé d'un lemme dont on sait qu'il désambiguïsera :

```
>0 cat:ponctuation
=0 ca:pronom
=0 cat
```

3) Catégorie *ad hoc* de délimiteurs de mots.

Si dans la langue étudiée il existe des symboles ou combinaisons de symboles qui avec certitude ne peuvent jamais apparaître à l'intérieur d'un mot ou lemme, il est intéressant de les regrouper. Voir par exemple la liste des mots de la catégorie "coupureDeMot" de LEXS-J-β3.X.X .

L.6. Morphologie et syntaxe ont même statut - Considérations générales.

SAGACE ne fait pas d'analyse morphologique. Les composants morphologiques de base sont lemmatisés au même titre que les composants syntaxiques. Ainsi, un verbe conjugué comme

飲みたくなかった

est traditionnellement tenu pour un mot fléchi de la forme (en gros) :

飲 radical + み base + た suffixe flexionnel : désidératif + くな suf.flex.: négation + かった suf.flex.:parfait

Dans le lexique utilisé par SAGACE, on se contente de lemmatiser le radical d'une part, et la chaîne de suffixes lexionnel avec la base d'autre part. Ainsi, le verbe est considéré comme la concaténation de deux lemmes :

飲 + みたくなかった

Ce découpage est motivé dans la présentation de LEXS-J-3.X.X.

L.7. Morphologie et syntaxe ont même statut - applications

Chercher les phrases comprenant un sujet (nom commun + particule sujet), et en fin de phrase (c'est à dire juste devant le point final de la phrase) un groupe numéral, immédiatement suivi d'un verbe au parfait (trait "conjTa").

```
>0 cat:nomCommun
=0 が
>0 cat:chiffre
=0 cat:specNum
=0 cat:vRad
=0 cat:conjTa
=0 。
```

Pour imposer que le tout soit à la fin de la phrase, on contraint le verbe à être contigu au point final de la phrase.

L.8. Chercher un lemme inconnu

Mettons que nous voulions constituer une liste expérimentale de spécifiques numériques à partir de la recherche ci-dessus. Il suffit de faire une recherche de collocations en n'enregistrant que le caractère trouvé. On disposera en sortie d'une liste de caractères provisoirement catégorisés " spécifiques numériques ". Ce fichier est ensuite intégré au lexique.

M. Tutoriel SAGACE 3.3.X et supérieur, résultats KWIC au format HTML

Il est désormais possible d'enregistrer le concordancier au format KWIC dans un fichier au format HTML. Voici un exemple.

La recherche est la suivante :

```
Travail: exemple
Resu:/home/user/sagace/resu/xx.html
RepertoirLexic: /home/user/sagace/lexic/
ScriptCorpus:/home/user/sagace/corpus/scriptCorpus.txt
NomDuGroupe:corpusEssais
Arret:"。"
Langue: japonais
Affiche le nom du fichier d'origine
Nombre max des exemples:40
Numerote les segments
Sauvegarde au format KWIC avant=20 apres=20:
Afficher le KWIC en HTML: "entete.txt"

Reqet:
>0 cat:particule
=0 cat:nom /-affich:trait:lemme /-KWIC1
=0 cat:particule
```

Ici, le fichier de format est celui donné par défaut. Il était aussi possible de ne pas mentionner le nom du fichier :

```
Afficher le KWIC en HTML:
```

Voici l'entête et en particulier le format d'affichage :

```
<html><head>
      <meta http-equiv="content-type" content="text/html; charset=sjis">

      <STYLE type="text/css">
            <!--
            table {table-layout:fixed ; }
            td.c0 {text-align: left;
                  font-size : 13px ;
                  font-family : arial, helvetica , sans-serif ;}
            td.c1 {
                  text-align: left;
                  font-size : 13px ;
                  font-family : arial, helvetica , sans-serif ;
                  }
            td.c2  {
                  text-align: right;
                  font-size : 13px ;
                  font-family : arial, helvetica , sans-serif ;
                  }
            td.c3  {
                  text-align: center;
                  background-color: rgb(204, 255, 255);
                  font-size : 13px ;
                  font-family : arial, helvetica , sans-serif ;
                  }
            td.c4  {
```

```
                    text-align: left;
                    font-size : 13px ;
                    font-family : arial, helvetica , sans-serif ;
                    }
              td.c5  {
                    text-align: left;
                    font-size : 12px ;
                    font-family : arial, helvetica , sans-serif ;
                    }
              -->
       </STYLE>
</head>
```

Le résultat obtenu est :

1	ずれにしても必ずその	身	を失うべき筈の捕虜が	doreikonjo_ron1.txt-
2	その身を失うべき筈の	捕虜	が、生命だけは助けら	doreikonjo_ron1.txt-
3	にして言えば、これが	原始時代	における奴隷の起源の	doreikonjo_ron1.txt-
4	始時代における奴隷の	起源	のもっとも重要なるも	doreikonjo_ron1.txt-
5	のもっとも重要なるも	の	である。	doreikonjo_ron1.txt-
6	か	つて	は敵を捕えればすぐさ	doreikonjo_ron1.txt-
7	捕えればすぐさまその	肉	を食らった赤色人種も	doreikonjo_ron1.txt-
8	後にはしばらくこれを	生	かして置いて、部落中	doreikonjo_ron1.txt-
9	》にしたり、あるいは	手足	の指を一本一本切り放	doreikonjo_ron1.txt-
10	あるいは灼熱した鉄の	棒	をもって焼き焦したり	doreikonjo_ron1.txt-
11	き焦したり、あるいは	小刀	をもって切り刻んだり	doreikonjo_ron1.txt-
12	て、その残忍な復讐の	快楽	を貪った。	doreikonjo_ron1.txt-
13	けれどもやがて農業の	発達	は、まだ多少食人の風	doreikonjo_ron1.txt-
14	達は、まだ多少食人の	風	の残っていた、蛮人の	doreikonjo_ron1.txt-
15	の残っていた、蛮人の	こ	の快楽を奪ってしまっ	doreikonjo_ron1.txt-
16	虜は駄獣として農業の	苦役	に使われた。	doreikonjo_ron1.txt-
17	また等しくこの	農業	の発達とともに、土地	doreikonjo_ron1.txt-
18	しくこの農業の発達と	とも	に、土地私有の制度が	doreikonjo_ron1.txt-
19	とともに、土地私有の	制度	が起った。	doreikonjo_ron1.txt-
20	そしてこの	こと	もまた、奴隷の起源の	doreikonjo_ron1.txt-

Pour exploiter ce fichier avec un tableur, il suffit d'enregistrer ce fichier n'importe où et selon le tableur, soit d'*importer* ce fichier HTML soit de l'*insérer* .

N. Versions

Numérotation

Numérotation à trois chiffres sagace-vX.X.X:
- Le troisième pour les corrections d'erreurs ou modif du manuel;
- Le second pour les modifs mineures, essentiellement des ajouts de fonctionnalités.
- Le premier pour les progrès majeurs.

Juillet 2012/07/10, sagace-v4.2.0
- Permet d'étudier plusieurs structures avec une seule requête.
- Ajout de l'option de contrôle sur le nombre maximum de cas trouvés dans un segment.
- Correction d'un bug important qui faussait les comptages pour les structures discontinues car toutes les structures possibles n'étaient pas prises en compte.
- Nombreuses mises au propres et simplifications du code.
- Modifie la lecture de la ligne de commande de sorte a désambiguïser
- Ajout d'une option permettant de changer l'ordre d'affichage des composants dans le résultat.
- Ajout d'un nouveau type d'affichage des résultats pour l'extraction d'exemples
- Ajout de l'option d'insertion de marque de début de segment.
- Plusieurs corrections dans le manuel.

Février 2011 , sagace-4.1.0
- Ne gère plus que les options d'affichage /-affich:"..." et /-affich:trait:..., /-compte , /-KWIC1, /-comptageRef.
- Assouplit l'écriture des options avec valeur numériques dans la deuxième partie de l'entête de la requête (modifications dans optionGenerales.c).
- Debogage dans work_f.c pour les problèmes de longueur maximales des chaînes manipulées. Réécriture du code pour certaines parties de work_f.c .
- Plusieurs corrections dans le manuel.

Janvier 2010, sagace-v4.0.0
Modification majeure :
- abandon des options spécifiques à une langue (en particulier la conjugaison à la forme neutre des radicaux fléchissables en japonais)
- uniformisation des commandes d'affichage des résultats en exploitant le système de traits
Plusieurs petites modifications d'enrichissement
- le segment peut désormais être délimité par plus d'une chaîne
- correction de bug : l'affichage KWIC rencontrait des problèmes lorsque le segment contenait un saut de ligne.
- possibilité de commenter plusieurs lignes avec la balise <commentaire>
- désormais, dans les comptages, les nombres d'occurrences sont distingués du reste par un encadrement : `[occ:X]` avec X le nombre d'occurrences du morphème qui précède
- correction d'un bug concernant l'ordonnancement des collocations + possibilité d'afficher dans l'ordre croissant et décroissant.
- la requête est désormais enlistée dans un tableau (100x300) et les lignes sont d'emblées mises au propres ; cela évite de multiplier les accès fichier et permet de simplifier le traitement de cette requête : vérification syntaxique, traitement des options, traitement de la chaîne à chercher.
- mise au propre de mk_list0.c qui contenait pas mal de fonctions inutiles.
- ajout fonction printf() dans fonctionsPratiqes.c pour le debugage.
- ajout possibilité de conversion à la volée des caractères latins et chiffres encodés sur deux octets en caractères codés sur un octet.
- modification du manuel

Janvier 2010, sagace-v3.3.0
- affichage au format HTML de la sortie KWIC
- modification du manuel en conséquence
- plusieurs correction du manuel («`Affiche le nom du fichier d'origine`»)

Septembre 2009, sagace-v3.2.1
- correction erreur fatale, suite à modification de notation des contraintes sur environnement.
- ajout d'un contrôle sur la bonne utilisation de l'option KWIC1.
- dans l'ensemble des fichiers, élimination de certains commentaires
- modification de ce manuel

Septembre 2009, sagace-v3.2.0
- enrichissement des modes d'affichage des résultats (options `-affich:_`)
- introduction du format KWIC pour l'affichage des exemples
- introduction du relevé des chaînes inconnues.
- pas mal de mise au propre du code dont réécriture simplifiée pour les conditions sur la présence de sous-chaîne et autres
- modification du mode de notation des contraintes sur l'environnement, désormais exclusivement exprimé par des traits.
- modifications du manuel.

Mars 2009, sagace-v3.1.0
- longueur des constantes propositionnelles augmenté à 500 octets.
- correction dans la manipulation de fichier (condition pour la recherche d'un mot strictement égal).
- ajout de la fonction d'insertion de nouvelle entrée.
- `saut_a_la_ligne` comme condition d'arrêt.

Janvier 2009, sagace-v3.0.1
- correction bug pour gestion de l'option "espaceDevant" ; nettoyage du code dans li_reqet.c relatif à la valuation des options d'affichage.

Janvier 2009, sagace-v3.0.0
- Introduction d'un langage de requête à base de formule des propositions.
- Refonte de l'organisation des fichiers sources

Juin 2008, sagace-v2.3.0
Cette version n'a pas été publiée ; elle ne présentait que des modifications mineures ;

Avril 2008, sagace-v2.2.0
- Ajout des fichiers listeCat.c et listeTraits.c pour lister les traits et les catégories du lexique indiqué dans le fichier de requête.
- Ajout du fichier cptNbEntrees.c pour compter le nombre total d'entrées.
- Savegarde automatique des résultats de la recherche statistique (c'est désormais une option au lieu d'être imposé) ; désactivée, l'accès au disque est réduit.

Février 2008, sagace-v2.1.0
- Modification de cat_val : l'entrée lexicale devient elle-même un trait
- Correction dans le manuel (dont passage relatif à l'utilisation de l'option «modification») et indication sur le trait lexical. Modification des données relatives à la ligne de commande.
- Modification de work_s.c et de l'option «modification» pour tenir compte de la conjugaison en `ゝ` en japonais.

Janvier 2008, sagace-v2.0.0
L'organisation du dictionnaire et sa gestion sont profondément modifiés. La catégorisation se fait désormais exclusivement par système de traits, et ne supporte plus l'ancien mode de catégorisation.
- Modification lit_reqet.c et work_f2.c
- Modification du présent manuel.

Janvier 2008, sagace-v1.3.0 ; non publié
- Ajout d'un deuxième type de notation des sous-catégories, par traits
- Modification lit_reqet.c et work_f2.c
- Modification du présent manuel.

Juillet 2007, sagace-v1.2.0
Intégration de l'option de contrôle sur les chaînes en double dans le corpus.
- Modification lit_reqet.c et work_f2.c
- Modification du présent manuel.

Juin 2007, sagace-v1.1.0
- Modification commentaire fichier inioptio.c
- Introduction de nouvelle fonctionnalité : possibilité de choisir le fichier de requête par commande en ligne.
- Modification du présent manuel.

Octobre 2006, sagace-v1.0.3 (version non diffusée)
- Modification du présent manuel, regroupé en un seul fichier (un au format odt (openoffice) l'autre au format pdf).

Février 2006, sagace-v1.0.2
- Correction dans un *for* de li_reqet.c.

Janvier 2006, sagace-v1.0.1
- Correction d'erreurs dans quelques sources, à l'origine d'une erreur de compilation.
- L'éxécutable change de nom : `sagace` au lieu de `reqet`
- Le deuxième nombre dans les lignes de requête (reliqat de la version *originelle*) disparaît.
- Sauvegarde des statistiques: désormais, une sauvegarde est automatiquement lancée lorsque le fichier en cours d'étude est terminé; sans cela, aucune sauvegarde si le quota n'était pas atteint et du coup, perte des quelques résultats.
- Modifications du manuel.

Janvier 2005, sagace-v1.0
- sagace-v1.0

O. Liste des défauts connus

Janvier 2012

- Bug : dépassement de mémoire (prob malloc) avec le script :
    Sauvegarde au format KWIC sans repetition avant=30 apres=31
    Reqet:
    >0 距離 /-affich:"距離" /-KWIC1
=0 cat:chiffre
- pas de vérification du bon guillemage dans la requête
- pas de vérification du bon numérotage des composant à dénombrer.
- bug avec des descriptions complexes

P. Bibliographie

Parutions (présentation, exemples d'applications) en relation avec SAGACE.

[1] Blin R., 2011, *Fréquence des occurrences de noms communs usuels dans le corpus statistique japonais BCCWJ - résultats complets* .

[2] Blin R., 2011 (à paraître), *Contribution de la linguistique de corpus à l'enseignement du japonais* , Actes du colloque de la SFEJ 2010, Piquier.

[3] Blin R., 2011 (à paraître), Enrichir le dictionnaire avec des données lexicométriques obtenues par analyse de corpus - cas du japonais- , Actes du colloque JEC 2010, INALCO.

[4] Blin R., 2009, SAGACE-v3.2 ; Analyseur de corpus pour langues non flexionnelles, TALN 2009, ATALA.

Index