KNG: un outil pour l'écriture facile de cascades de transducteurs

François Barthélemy[1, 2]
(1) INRIA – Alpage, domaine de Voluceau 78153 Rocquencourt
(2) CNAM – Cedric, 292 rue Saint-Martin, 75003 Paris
francois.barthelemy@inria.fr

Résumé. Cet article présente une bibliothèque python appelée KNG permettant d'écrire facilement des automates et transducteurs finis. Grâce à une gestion soigneuse des codages et des entrées-sorties, cette bibliothèque permet de réaliser une cascade de transducteurs au moyen de tubes unix reliant des scripts python.

Abstract. This paper presents a Python library called KNG which provides facilities to write easily finite state automata and transducers. Through careful management of encodings and input/output, a transducer cascade may be implented using unix pipes connecting python scripts.

Mots-clés : Machines finies à états, cascade de transducteur, expression régulière.

Keywords: Finite State Machines, Transducer Cascade, Regular Expression.

1 Chaînes de traitements et transducteurs finis

Les machines finies à états, c'est à dire les automates et transducteurs finis, ont conquis des domaines d'application pour lesquels ils sont bien adaptés du fait de leur efficacité. Dans le domaine du TAL, il s'agit entre autres de la représentation de lexiques, de l'analyse morpho-lexicale, de l'étiquetage syntaxique, de diverses formes de recherche de motifs ainsi que de l'approximation de tâches pour lesquels elles ne sont pas assez puissantes comme l'analyse syntaxique.

D'une part des outils ont vu le jour pour faciliter la description de machines finies : ce sont les expressions régulières et leurs extensions, notamment les règles de réécriture contextuelles. Il faut noter d'ailleurs que ce type de règles est utilisé spécifiquement pour le TAL. D'autres part, les expressions régulières sont devenues un mécanisme de traitement des chaînes de caractères important dans de nombreux langages de programmation. Mais les deux mondes sont restés jusqu'à ce jour relativement distincts : les systèmes dédiés aux machines finies ne communiquent pas facilement avec d'autres applications alors que les langages comme perl ne proposent pas d'opérations combinant deux machines finies ou ayant une machine finie comme résultat.

Pour bien comprendre la différence, prenons l'exemple de la substitution de sous-chaînes. Il est facile d'écrire un script Perl de 5 lignes qui prend des chaînes de caractères et remplace dans ces chaînes toutes les occurrences d'une sous-chaîne s1 par une autre sous-chaîne s2. Un transducteur fini peut faire le même travail sur les chaînes, mais il peut aussi s'appliquer de façon plus générale à n'importe quel ensemble régulier de chaînes. On peut appliquer un transducteur fini à un langage régulier et obtenir comme résultat un autre ensemble régulier. C'est plus efficace que de procéder chaîne par chaîne : d'une part cela permet de traiter en temps fini un ensemble infini de chaîne, s'il est régulier. D'autre part cela permet un gain de complexité grâce au partage des calculs : on passe d'une complexité exponentielle à une complexité quadratique.

Plus spécifiquement pour le TAL, l'utilisation de machines finies, éventuellement avec une restriction aux machines acycliques, permet de représenter l'ambiguïté de certains composants et de considérer en parallèle plusieurs analyses alternatives. Cette aptitude est utilisée par certaines techniques de correction d'erreurs où plusieurs corrections peuvent être proposées.

L'utilisation de poids dans des machines finies pondérées permet de gérer l'ambiguïté : à certains stades, l'ambiguïté peut être résolue ou diminuée au moyen d'un algorithme de détermination du meilleur ou des n meilleurs chemins.

Dans cet article, nous décrivons une bibliothèque python permettant d'écrire facilement des machines finies pondérées avec un haut degré d'intégration au langage hôte. Cette bibliothèque permet non seulement de créer des automates et transducteurs, mais elle permet également de les importer et exporter facilement sous différents formats. Cela permet la

réalisation de cascades de transducteurs sous la forme de tubes unix (pipes) de scripts pythons.

Les principales caractéristiques de la bibliothèque seront exposées : la représentation des machines ; les opérations ; la représentation des caractères et leur codage ; les entrées-sorties ; la compilation statique dans un langage interprété. Nous terminerons avec une rapide comparaison avec d'autres langages de machines finies.

2 KNG et les machines finies

KNG est un module python qui offre un certain nombre de fonctions et de classes permettant la représentation et la manipulation de machines finies pondérées. En premier lieu, il y a deux classes différentes pour les automates (KNG.automaton) et les transducteurs (KNG.transducer). Une machine finie pour KNG est un objet instance de l'une de ces deux classes. Ces objets peuvent être stockés dans des variables python, passés en paramètre à des fonctions et des méthodes. Ce sont donc des valeurs python au plein sens du terme.

La construction des objets peut se faire de deux façon différentes : directement par le constructeur de la classe, ou de façon incrémentale en ajoutant successivement des éléments au moyen de méthodes. La construction directe se fait au moyen d'une expression régulière représentée par une chaîne de caractère python. Pour la construction incrémentale, une forme mutable de machine initialement vide est créée par le constructeur, puis des éléments sont ajoutés par des appels de méthodes. Ces éléments sont soit des noeuds et arcs, soit des chaînes ou paires de chaînes appartenant à l'ensemble régulier que la machine représente.

```
# construction directe (alp : alphabet défini préalablement)
ex1=KNG.automaton(alp,'(a|b|c)(d|e)')
# construction incrémentale 1
ex2==KNG.automaton(alp)
ex2.add_path('ad')
ex2.add_path('ae')
# construction incrémentale 2
ex3==KNG.automaton(alp)
st1=ex3.add_state()
st2=ex3.add_state()
ex3.add_arc(st1,st2,'a')
ex3.set_initial(st1)
ex3.set_final(st2)
```

Pour la représentation interne des machines, KNG utilise la bibliothèque OpenFst de google, qui est un outil écrit en C++. L'accès à cette bibliothèque est transparent pour l'utilisateur qui ne voit que le module python KNG. Le type de poids utilisé est le type standard des poids d'OpenFst, à savoir le semi-anneau tropical.

Les opérations sur automates et transducteurs sont des méthodes des classes considérées. Dans les deux classes, il y a l'union, la concaténation, la clôture étoile, la clôture plus et l'opération qui extrait les n chemins de poids minimal. Les opérations binaires (union et concaténation) ne sont définies que pour deux objets de la même classe, à savoir deux automates ou deux transducteurs. Les opérations spécifiques des automates sont l'intersection, la complémentation et la différence ensembliste. Les opérations spécifiques aux transducteurs sont la composition, l'inversion et l'application fonctionnelle (réécriture d'une entrée en une sortie).

```
# operation destructive, resultat dans ex1
ex1.concat(ex2)
# ex1 inchangé, résultat dans ex4
ex4=ex1.concat(ex2,'conservative')
# chaîne d'applications
ex1.concat(ex2).union(ex3).rm_epsilon()
```

Il y a ensuite des opérations de conversion d'une classe dans l'autre : le produit cartésien construit un transducteur à partir de deux automates. L'identité construit un transducteur à partir d'un seul automate. Les deux projections extraient l'un

ou l'autre côté d'un transducteur, ce qui donne un automate. La dernière opération de conversion est la plus complexe : c'est la *règle de réécriture contextuelle* (Kaplan & Kay, 1994). Elle prend quatre opérandes qui sont quatre automates finis représentant respectivement un motif à remplacer, un motif de remplacement, le contexte gauche et le contexte droit. Le résultat de l'opération est un transducteur fini qui réécrit en une seule fois toutes les occurrences du motif respectant les conditions de contexte.

Il existe trois opérations d'optimisation d'une machine finie : l'élimination des epsilon-transitions, la déterminisation et la minimisation.

KNG offre des mécanismes de conversion vers les chaînes de caractères de python. Enumérer les chaînes reconnues par un automate se fait au moyen d'un itérateur ce qui permet l'utilisation de la boucle for. Par exemple le code suivant permet d'afficher les chaînes d'un automate.

```
auto=KNG.automaton("(a|b|c)(d|e)")
for st in auto:
    print st
```

Si le langage de l'automate est infini, la boucle ne termine jamais. Dans l'autre sens, la conversion d'une séquence de chaînes en un automate se fait au moyen de la construction incrémentale de l'objet. Pour les transducteurs, ce ne sont pas des chaînes mais des paires de chaînes qui sont énumérées (type `tuple` de python).

3 Alphabets, codages, entrées/sorties

Par définition, les machines finies sont définies sur un ensemble fini de symboles appelé un alphabet. Les éléments de cet ensemble peuvent être des caractères, c'est à dire des composants de base des chaînes de python, mais ils peuvent également être d'une autre nature. Les caractères peuvent être représentés en machine selon différents codages : iso-8859, unicode, utf-8, etc.

Un exemple de symboles d'ensemble finis qui ne sont pas des caractères et qui sont couramment utilisés en linguistique dans des machines finies sont les étiquettes syntaxiques telles que Prep, Det, Adv, etc. En KNG, ces symboles sont identifiés au moyen d'un identificateur qui commence par une lettre et se poursuit par une suite quelconque de lettres, de chiffres et de soulignés. Dans la suite nous les appellerons *symboles multicaractères*.

En interne, les machines finies utilisent une représentation des caractères sous forme de nombres entiers. Les caractères peuvent être représentés en utilisant un codage au choix parmi quelques codages prédéfinis et l'utilisateur peut également spécifier sa propre table de conversion. Pour ce qui est des symboles multicaractères, KNG leur octroie un code dans les plans privés d'unicode sauf si l'utilisateur spécifie lui-même le codage à employer. Dans les expressions régulières, les symboles multicaractères sont entourés de < et > (par exemple `<Prep>`). Un alphabet spécifie le codage utilisé en interne dans la bibliothèque. Il n'exclut pas l'utilisation d'autres codages pour les entrées-sorties.

Les alphabets de KNG sont structurés en classes de symboles qui sont simplement des ensembles finis, pas nécéssairement disjoints, de symboles. Ces classes ont des noms qui peuvent être utilisés dans les expressions régulières pour dénoter la disjonction de leurs éléments. Les alphabets sont compilés en objets instances de la classe `KNG.alphabet`. Les constructeur des classes KNG.automaton et KNG.transducer prennent un objet alphabet en paramètre. Les opérations impliquant plusieurs machines ne sont définies que pour les machines utilisant le même alphabet.

```
alpstr = '''
using unicode;
class lettre="a..zéèêàùçôûâïëüÿ";
class sep=" -";
class chiffre="0123456789";
class trait="<masc><fem><sg><pl><ord><card>";
'''
alp = KNG.alphabet(alpstr)
```

KNG permet de faire des entrées-sorties de machines finies et d'alphabets au moyen de la notion de flot (stream). Cette

construction présente dans les langages de programmation modernes permet un abord abstrait de différent types d'entrées-sorties : entrée et sortie standards, fichiers et périphériques divers.

Un flot est une séquence de valeurs de même type. KNG propose des flots de machines finies et des flots d'alphabets. Ces flots peuvent utiliser différentes représentations textuelles ou binaires. Les flots binaires actuellement implémentés sont deux variantes du format binaire OpenFst : ce format sans modification, adapté pour communiquer avec les commandes OpenFst ; ce format enrichi avec des informations à propos de l'alphabet utilisé, adapté pour la communication entre scripts KNG. Les formes binaires, comme on peut s'y attendre, sont plus efficaces mais moins portables que les représentations textuelles.

Les formes textuelles sont plus adaptées pour communiquer avec des outils externes ou des êtres humains. Il y a d'abord les expressions régulières, limitées pour l'instant aux flots d'entrées. Un tel flot est une séquence d'expressions régulières. Chacune d'elle est compilée par l'opération de lecture en un objet KNG.automaton ou KNG.transducer. Nous souhaitons à l'avenir offrir également les expressions régulières pour les flots de sortie. Une autre forme textuelle acceptée par KNG est le format FSM qui décrit les états et les arcs de la machine. Ce format créé à l'origine pour FSM (Mohri *et al.*, 2002) est accepté par différents outils, notamment Openfst et Xfst. Nous voudrions ajouter ultérieurement des formats XML de représentation de machines finies tels que FSM-XML (Demaille *et al.*, 2009) ou SCXML (W3C, 2012).

Les formats textuels sont acceptés sous différents codages (utf-8 et quelques codages de la norme iso-8859) et ce de façon indépendante du codage utilisé en mémoire pour les objets créés. Par exemple, KNG peut lire un flot codé en iso-8859-1 pour créer un objet utilisant le codage unicode.

Les flots dans KNG sont implémentés par des classes. Généralement, l'utilisateur ne crée pas directement des objets de ces classes mais utilise une fonction `open` qui construit et renvoie un objet d'une classe ou d'une autre en fonction de la valeur des paramètres. KNG s'inspire ici de la gestion des flots par python. La fin de flot se traduit par la levée d'une exception. Les flots en écriture offrent deux méthodes : l'écriture d'une valeur (write) et la fermeture du flot (close). Les flots de lecture offrent une méthode de lecture (read) qui renvoie une machine finie ou un alphabet et des itérateurs permettant d'utiliser directement un flot dans une instruction `for`.

```
# flot d'entrée binaire dans un fichier, codage utf-8
input_stream_1=open('converter.bin','r','machine','utf-8',alp)
trans=input_stream_1.read()
# flot d'entrée textuel expression régulière, entrée standard
input_stream_2=KNG.open(sys.stdin,'r','machine','utf-8','regexp',alp)
# flot de sortie textuel fsm, sortie standard
output_stream=KNG.open(sys.stdout,'w','auto','utf-8','fsm')
# iterateur sur input_stream_2, aut est une machine finie
for aut in input_stream_2:
   output_stream.write(trans.apply(aut,'lr'))
output_stream.close()
```

La méthode `apply` utilisée dans la boucle est l'application fonctionnelle du transducteur. A chaque tour de boucle, une expression régulière lue sur l'entrée standard est compilée en une machine finie désignée par le nom `aut`, utilisée comme entrée pour le transducteur `trans`, et la sortie du transducteur est écrite sur la sortie standard au format `fsm`.

4 Compilation et langage interprété

Python est un langage interprété. Cela signifie qu'une expression régulière apparaissant dans un programme est recompilée à chaque exécution du programme, même si elle ne change pas d'une exécution à l'autre. Ce n'est clairement pas ce que désire une personne écrivant une cascade de transducteur : elle souhaite que tout le travail faisable statiquement ne soit fait qu'une seule fois. Sur ce point, KNG s'est inspiré de ce que fait PLY, une implémentation en python de LEX et YACC. Une grammaire PLY est un programme python. A la première exécution de ce programme, PLY compile la grammaire et crée des tables sauvegardées dans des fichiers. Aux exécutions suivantes, PLY vérifie que la grammaire est inchangée. Si c'est le cas, les fichiers sont relus. Sinon, la grammaire est recompilée.

KNG procède de façon analogue. Par une analyse de la façon dont une machine est créée, KNG détecte si c'est une construction statique ou dynamique. Par exemple, une expression régulière donnée sous forme d'une chaîne de caractère

constante est statique. Une expression régulière lue sur un flot donne naissance à un objet dynamique. En cas de doute, une machine est considérée comme dynamique.

KNG sauvegarde dans des fichiers les machines statiques. Lors d'une nouvelle exécution du programme, les dates du fichier source python et de la machine sauvegardée sont comparées pour déterminer si la description de la machine a pu changer entre temps. Si ce n'est pas le cas, le fichier de sauvegarde est relu. Par ailleurs, dans ce processus de sauvegarde des objets statiques, les dépendances entre machines sont représentées et sauvegardées. Il se peut que la définition d'une machine soit inchangée mais qu'un élément précurseur ait évolué rendant nécessaire un recalcul. Par exemple, si une machine A est obtenue en réalisant une clôture étoile d'une machine B, tout changement de B oblige à recalculer A. C'est pour cela qu'une relation de dépendance entre composants doit être maintenue et mémorisée.

Dans son état actuel, la gestion des dépendances par KNG est assez rudimentaire : les dépendances sont enregistrées entre fichiers sur la base de la date de dernière modification. La détermination des éléments statiques est également rudimentaire. Des sous-composants statiques de machines dynamiques pourraient être sauvegardés.

5 Relations entre KNG et Python

KNG n'utilise pas d'espace de noms spécifique. Pour nommer une machine, on peut utiliser tout simplement une variable python qui contiendra une référence à l'objet implémentant la machine. C'est dans ce cas l'espace de noms du programme utilisateur qui est utilisé. Une autre solution est de stocker la paire (nom,valeur) dans un dictionnaire (tableau associatif) Python. Ce qui est moins simple et moins courant, c'est que ce nom peut être utilisé à l'intérieur d'une expression régulière. Pour distinguer le nom d'une machine préexistante `m` d'une séquence de caractères, le nom doit être encadrés par deux symboles ‘ (bacquote) : `` `m` `` est un nom, `m` le caractère m.

La variable n'appartient de toute façon pas à l'espace de nom du module KNG qui n'est ni celui du programme utilisateur, ni le dictionnaire utilisé pour stocker les machines finies. Si une expression régulière contient des références à des machines préexistantes, l'espace de nom à utiliser doit être passé en paramètre au compilateur KNG.

```
auto1=KNG.automaton(alp,'cd')
# auto1 variable python
# espace de nom: celui du script python
auto3=KNG.automaton(alp,'ab|`auto1`|ef',globals())
# le troisième paramètre est l'espace de nom
```

L'utilisation de l'espace de noms du programme utilisateur permet de structurer les définitions de machines finies au moyen de paquetages Python. Par exemple, si l'on définit une machine `m1` dans un paquetage `A`, on pourra la référencer dans une expression régulière d'un programme qui importe `A` au moyen du nom qualifié `A.m1`. On peut utiliser les mécanismes de visibilité Python (à savoir la variable __all__ du fichier __init__.py) pour spécifier si une machine est visible ou non à l'extérieur d'un paquetage.

Les constructions de programmation Python (boucles, exceptions, classes, fonctions) peuvent être utilisées pour une application en KNG. Par exemple, une fonction peut être définie qui prend en paramètre un automate fini, réalise l'application fonctionnelle de plusieurs transducteurs en cascade, puis optimise le résultat de cette cascade au moyen d'élimination d'epsilon-transitions, déterminisation et minimisation et enfin renvoie l'automate résultant de cette séquence.

Comme nous l'avons déjà mentionné, KNG procure des itérateurs permettant de parcourir des ensembles de valeurs au moyen d'un boucle for. Ces itérateurs sont disponibles sur les machines (itération sur les chaînes du langage d'un automate, itération sur les paires de la relation d'un transducteur) et sur les flots d'entrées (itération sur les alphabets ou les machines selon le type du flux).

6 Comparaison avec d'autres outils

Il est impossible de faire une comparaison exhaustive avec tous les langages et systèmes existants qui sont fort nombreux. Nous allons prendre quelques exemples représentatifs des systèmes utilisés pour le TAL.

Commençons par la façon dont Perl utilise les expressions régulières. Les expressions régulières de Perl sont utilisées pour spécifier deux opérateurs et une fonction de traitement de chaînes : la reconnaissance de motif, la substitution et le découpage respectivement notés `m//`, `s///` et `split`. Il n'y a pas véritablement d'opérateur permettant de combiner des expressions régulières. Tout au plus une expression régulière quotée peut-elle être utilisée comme constituant d'une autre expression régulière. Les opérateurs de reconnaissance et de substitution ne s'appliquent qu'à des chaînes et leur résultat est une chaîne.

L'opérateur de substitution n'est pas un transducteur fini : en effet la chaîne qui remplace la sous-chaîne appariée au motif de la substitution est calculée dynamiquement, une fois liées les variables du motif. Le calcul peut d'ailleurs inclure l'évaluation d'une expression telle que par exemple un appel de fonction. La substitution est de ce fait beaucoup plus puissante qu'un transducteur fini : elle a la puissance de la machine de Turing. En contre-partie, elle est moins susceptible de précompilation et ne factorise pas les calculs.

Il existe des systèmes qui implémentent de façon exemplaire les machines finies et leurs opérations mais sont très sommaires du point de vue des structures de programmation. C'est le cas de Xfst (Xerox Finite State Tool). Xfst est utilisé surtout pour la morphologie et la phonologie (Beesley & Karttunen, 2003). Il dispose des expressions régulières habituelles étendues avec les règles de réécriture contextuelles. Des descriptions morphologiques ou phonologiques très élégantes ont été données sous forme de cascades de règles contextuelles, compilées sous forme d'une cascade de transducteurs. Mais le langage proposé est très pauvre : il n'y a pas réellement de variables, pas de modularité, pas de fonction, pas de boucle ni de conditionnelle. Pratiquement, on ne dispose que de la séquence qui permet d'exécuter des opérations l'une après l'autre.

D'autres système font le choix de se présenter comme KNG sous la forme d'une bibliothèque, mais dans la plupart des cas, ce sont des bibliothèque C++. Citons notamment `OpenFst` et Stuttgart Finite State Transducer tools (SFST) (Schmid, 2005). Les structures de programmation sont riches, mais au prix d'une complexité du langage hôte qui rend le développement plus difficile.

PyOpenFst [1] est comme KNG une couche encapsulant OpenFst dans un module Python. Comme OpenFst, il n'implémente pas les expressions régulières ni a fortiori les règles de réécriture contextuelle. Il n'offre pas non plus la même variété d'entrées-sorties.

KNG est un outil opérationnel mais encore expérimental. Au moment où cet article est écrit, il est utilisé pour réécrire partiellement SxPipe (Sagot & Boullier, 2005), une chaîne de traitement morpho-lexical. Dans cette chaîne, des composants KNG sont utilisés dans une cascade comportant également des scripts Perl et des programmes en C.

Nous prévoyons de diffuser prochainement KNG sous licence LGPL.

Références

BEESLEY K. R. & KARTTUNEN L. (2003). *Finite State Morphology*. CSLI Publications.

DEMAILLE A., DURET-LUTZ A., LESAINT F., LOMBARDY S., SAKAROVITCH J. & TERRONES F. (2009). An xml format proposal for the description of weighted automata, transducers and regular expressions. In *Proceedings of the 2009 Conference on Finite-State Methods and Natural Language Processing : Post-proceedings of the 7th International Workshop FSMNLP 2008*, p. 199–206, Amsterdam, The Netherlands.

KAPLAN R. M. & KAY M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, **20 :3**, 331–378.

MOHRI M., PEREIRA F. C. N. & RILEY M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, **16**(1), 69–88.

SAGOT B. & BOULLIER P. (2005). From Raw Corpus to Word Lattices : Robust Pre-parsing Processing with SxPipe. *Archives of Control Sciences*, **15**(4), 653–662.

SCHMID H. (2005). A programming language for finite state transducers. In A. YLI-JYRÄ, L. KARTTUNEN & J. KARHUMÄKI, Eds., *Finite-State Methods and Natural Language Processing*, volume 4002 of *Lecture Notes in Computer Science*, p. 308–309, Helsinki, Finland.

W3C (2012). State chart xml (scxml) : State machine notation for control abstraction w3c working draft 16 february 2012.

1. http ://code.google.com/p/pyopenfst/