Utilisation d'outils de TAL pour la compréhension des spécifications de validation de données

Arthur Remaud

Actes de la 28e Conférence sur le Traitement Automatique des Langues Naturelles
Lille, France, 28 juin au 2 juillet 2021

Utilisation d'outils de TAL pour la compréhension de spécifications de validation de données

Arthur Remaud [1,2,3]
(1) Clearsy, 320 Av. Archimède – Les Pléiades III, 13100 Aix-en-Provence, France
(2) Samovar, 9 rue Charles Fourier, 91011 Evry, France
(3) LISN, Campus Universitaire bâtiment 507, Rue du Belvédère, 91405 Orsay, France
arthur.remaud@clearsy.com

RÉSUMÉ

La validation de données consiste à vérifier formellement la cohérence de données utilisées en entrée de systèmes critiques. L'essentiel du travail des ingénieurs consiste donc à traduire une spécification, écrite en langage naturel, en un ensemble de règles formelles permettant l'automatisation de la vérification. Notre objectif à long terme est d'automatiser complètement le processus de validation de données. Dans cet article, nous présentons une première étape et détaillons les différentes techniques de traitement automatique de la langue que nous avons déployées pour générer un squelette de règle formelle à partir d'une spécification textuelle. La particularité de ces spécifications est qu'elles peuvent contenir beaucoup d'informations implicites qui rendent difficile la tâche de traduction. D'autre part, le fait qu'il n'existe pas de grand corpus d'apprentissage disponible rend difficile l'emploi des méthodes d'apprentissage neuronal profond. Néanmoins des approches plus classiques à base de règles et de représentations symboliques permettent d'apporter un premier élément de réponse.

ABSTRACT

Use of NLP tools for automatic comprehension of data validation specifications

Data validation is the formal verification of data used in critical systems. The major part of the engineers'work consists in translating a natural language specification into a set of formal rules, allowing the automation of the verification. As a first step toward a full automation of this translation process, we detail in this article different natural language processing methods which we deployed to build a formal rule skeleton from a textual specification. The characteristics of these specifications are their use of implicit project information, which make harder the task of translation. Furthermore the absence of large corpus for machine learning, makes difficult the use of deep learning neuronal methods. However, more classic approaches based on rules and symbolic representations provide a first solution.

MOTS-CLÉS : TAL, extraction d'entités, relations entre entités, analyse syntaxique.

KEYWORDS: NLP, entities extraction, entity linking, syntactic analysis.

1 Introduction

La validation formelle de données consiste en la vérification formelle des données d'entrée et de fonctionnement de systèmes critiques. L'objectif est de vérifier la cohérence des données entre elles

afin de détecter de potentielles erreurs de relevé, voire du système (Lecomte & Mottin, 2016). Par exemple, un système de guidage de train doit nécessairement avoir en entrée un plan des voies identique à la réalité. Toutes ces données sont difficiles à vérifier manuellement de par leurs tailles et nombres importants, mais on peut vérifier automatiquement qu'elles suivent les spécifications des voies, par exemple *"A signal shall be at least 100 meters before a crossover"*.

Le travail des ingénieurs en validation de données consiste en grande partie à traduire en règles formelles les spécifications des clients concernant les propriétés à vérifier sur les données. La majorité de ces spécifications sont des phrases simples, que l'on voudrait pouvoir traiter automatiquement ou semi-automatiquement, afin que les ingénieurs consacrent plus de temps aux spécifications plus complexes qui demandent plus de réflexion.

Dans cette optique, l'utilisation d'outils d'analyse de la langue est nécessaire afin de pouvoir saisir le sens de la vérification à appliquer. La traduction de spécifications en règles formelles a déjà été exploré par le passé (Sadoun, 2014), mais avec des techniques de traitement de la langue éloignées de l'état de l'art. Actuellement, les programmes les plus performants pour la traduction, et l'extraction d'informations d'un texte reposent sur des techniques d'apprentissage automatique (Devlin *et al.*, 2018; Brown *et al.*, 2020), utilisant d'importants corpus de textes pour la phase d'apprentissage. Or, les différents projets industriels de validation de données ne comprennent, en général, guère plus de quelques centaines de spécifications au maximum, avec de grandes différences d'un projet à l'autre. A la connaissance des auteurs, il n'existe pas de corpus dans ce domaine permettant d'entraîner des réseaux de neurones profonds pour des tâches de traitement automatique de la langue. A cause de la grande complexité de la production de données annotées, et des données existantes en faible quantité, il convient d'expérimenter ces réseaux neuronaux pour des tâches plus simples que la traduction directe, et de combiner ces résultats pour parvenir à cet objectif.

Cet article montre différentes approches du traitement de la langue utilisées dans cet objectif industriel, afin d'établir un prototype pour traduire des spécifications, en partant d'un modèle de phrase précis (proche d'un langage contrôlé) pour ensuite l'étendre sur des formulations plus complexes et plus variées. La section 2 détaillera les techniques abordées, tout en prenant en compte la contrainte du corpus d'apprentissage afin de concevoir le prototype. Ensuite la section 3 détaillera un essai du prototype sur des spécifications issues d'un projet industriel. Enfin la conclusion apportera un bilan de ce travail et les perspectives d'avenir pour ce prototype.

2 Développement du prototype

A l'heure actuelle, les outils les plus performants en TAL se basent sur des réseaux de neurones profonds, dont la dernière couche est adaptée à la tâche voulue dans une étape d'apprentissage appelée *fine-tunning*, comme l'approche BERT (Devlin *et al.*, 2018) à base de « Transformer » (réseau neuronal profond avec une architecture encodeur-décodeur). Nous avons donc décidé d'utiliser des outils qui s'appuient sur ces algorithmes afin d'étudier leurs performances dans le cadre de la validation de données, limité notamment par le manque de données d'entraînement. Cette utilisation sera complétée par des des outils à base de règles pour compléter une traduction en règle formelle.

Parmi les informations que nous voulons extraire, nous voulons principalement savoir quelles données vont être analysées et quelle(s) vérification(s) est (sont) à appliquer. Pour cela, l'extraction d'entités et l'étiquetage de relations permet de ressortir les données et de voir comment elles s'articulent entre

elles, et l'analyse syntaxique indiquera la vérification à effectuer et l'ordre des données en paramètre.

2.1 Extraction d'entités

Chaque spécification produit une vérification à faire sur certaines données. On retrouve donc des noms d'objets, leurs paramètres ou simplement des valeurs que l'on aimerait détecter automatiquement. Dans notre cas, on recense quatre types d'entités à extraire :

— les classes, qui représentent les objets à analyser dans les données,
— les attributs, qui sont des paramètres des classes,
— les variables,
— les valeurs, comme des entiers, des chaînes de caractères, ou des valeurs qualitatives.

Parmi les approches récentes qui utilisent des plongements lexicaux de type BERT, nous avons utilisé l'application Bert-NER [1] qui a été choisie, car elle est facile d'utilisation en tant que bibliothèque, les phases de *fine-tuning* et d'utilisation d'un modèle entraîné étant bien découplées. Bien que ce programme n'utilise pas d'aide pour les informations implicites du projet, ses résultats sont satisfaisants, comme montré plus loin dans la section 3.2.1. Il existe différents modèles pré-entraînés, mais ils ne permettent pas de détecter les entités recherchées dans les spécifications, car le langage utilisé pour les spécifications et les concepts qui y sont référencés sont spécifiques au domaine. Pour y remédier, 140 spécifications d'un projet industriel ont été annotées afin de faire un *fine-tuning* spécialisé à notre tâche.

Associées aux entités, les relations qui les lient sont une source d'informations utiles pour la traduction de spécification dans un langage formel.

2.2 Étiquetage de relation entre entités

Les entités d'un texte sont très souvent liées entre elles. De nombreux travaux en TAL concernent l'identification des liaisons entre ces entités et la détermination du type des relations, comme par exemple entre un objet et son possesseur (Shi & Lin, 2019; Papanikolaou *et al.*, 2019).

Nous cherchons à étiqueter quatre types de relations entre les entités :

— les relations classe-attribut, déterminant à quelle classe est attaché un paramètre,
— les relations variable-valeur, indiquant à quelle variable est affectée une valeur,
— les relations variable-type, reliant une variable à son type lorsqu'il est indiqué dans le texte,
— les relations valeur-paramètre, indiquant qu'une valeur (données, variable) est conditionnée par un paramètre.

Ces relations permettent entre autre de faire de l'affectation de valeur à une variable, de faire du typage, ou d'éviter des ambiguïtés comme lorsque que plusieurs classes ont un attribut de même nom. Elles ne peuvent toutes être détectées par une analyse syntaxique car ces relations peuvent concernés des mots très éloignés dans une phrase.

Pour cet objectif, nous avons repris l'outil OpenNRE (Han *et al.*, 2019) sous licence MIT. A nouveau, il a fallu faire un *fine-tuning* pour les relations précédemment citées, avec le même jeu de données que celui utilisé pour l'extraction d'entités, étiqueté pour cette tâche. L'un des avantages de cet outil est qu'avec l'étiquetage de la relation, il y a un pourcentage de certitude de l'algorithme pour aider à

1. https://github.com/kamalkraj/BERT-NER

déterminer la véracité de l'information.

Pour compléter ces informations sur les données à valider, il faut aussi les informations sur le type de vérification demandée, que l'on peut trouver avec de l'analyse syntaxique.

2.3 Analyse de l'arbre syntaxique

Pour compléter l'étude des phrases, une première approche peut être l'analyse de sa structure et sa syntaxe. En regardant comment les mots s'articulent entre eux, on peut déjà apercevoir certaines relations sémantiques, notamment pour les phrases les plus basiques.

Dans beaucoup de règles, on observe que le groupe verbal indique la vérification à effectuer, par exemple *is greater than* ou *contains*. Pour chaque groupe verbal, on peut déterminer l'arité de l'opérateur associé ainsi que les rôles du sujet et des différents compléments, qui constituent ainsi les paramètres de la vérifications. Une liste de toutes les vérifications les plus courantes dans les spécifications a été construite pour pouvoir les repérer avec l'analyse syntaxique, avec à chaque fois les traductions en langage formel.

L'analyse des paramètres de la vérification repose sur les informations syntaxiques, mais aussi sur la présence des entités et leurs relations entre elles extraites par les outils des sections précédentes. Comme pour les groupes verbaux, une liste des principaux types de sujets et compléments est dressée, avec par exemple un sujet ne contenant qu'une donnée, ou un complément composé du mot *range* explicitant l'utilisation d'un intervalle, etc.

Pour l'analyse syntaxique, c'est la bibliothèque Python *Spacy* (Honnibal & Montani, 2017) qui a été retenue pour sa simplicité d'usage et ses résultats satisfaisants.

2.4 Assemblage dans un prototype

Toutes les techniques décrites précédemment sont assemblées dans un prototype qui permet d'analyser les spécifications textuelles et construire un squelette rédigé dans un langage naturel contraint, utilisé notamment pour que le client puisse valider la règle formelle facilement.

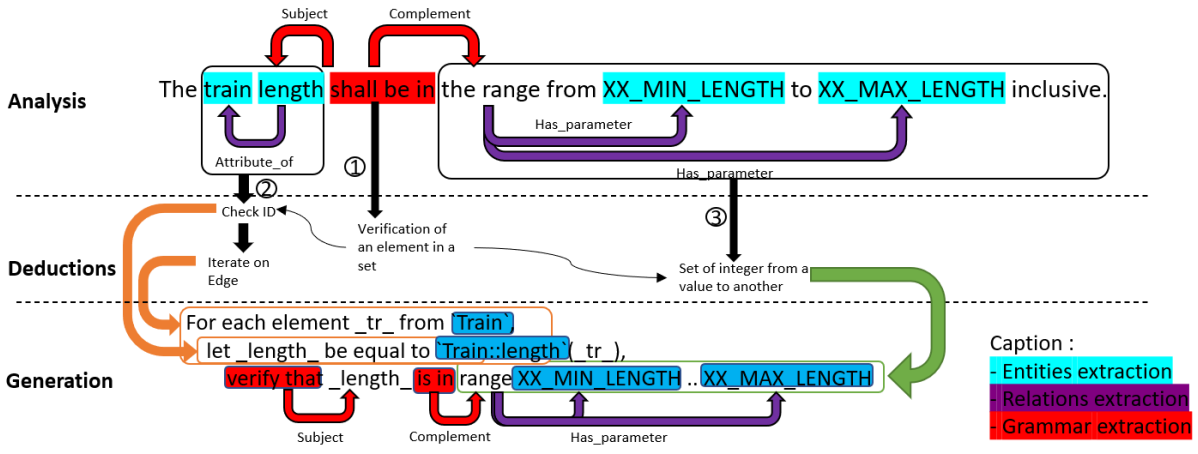

FIGURE 1 – Fonctionnement du prototype

Comme le montre la figure 1, nous utilisons l'analyse grammaticale afin d'extraire le groupe verbal (dans cet exemple *"shall be in"*) pour identifier la vérification (ici l'appartenance à un ensemble), ainsi que le sujet et les compléments pour avoir les paramètres. Dans ces paramètres, les programmes décrits dans les paragraphes précédents aidés des données du projet extraient les données (ici la taille du train), les variables (comme l'intervalle) et les valeurs (ici les constantes *XX_MIN_LENGTH* et *XX_MAX_LENGTH*), ainsi que les relations entre elles (respectivement *class-attribute* et *value-parameter*).

Chaque élément est traduit par une règle préétablie en langage naturel contraint, et le tout est assemblé en suivant l'architecture grammaticale afin d'obtenir la règle finale. Ainsi la règle itère sur la donnée indiquée par le sujet, puis s'assure que cette donnée est bien dans un intervalle entre deux constantes.

3 Utilisation du prototype sur un projet concret de validation de données

Les différentes techniques d'analyse de spécifications présentées dans la section précédente ont été testées sur un projet concret de validation de données de l'entreprise Clearsy [2]. L'objectif n'est pas de tout traduire parfaitement, mais de quantifier le nombre de spécifications pouvant être analysées entièrement par ce prototype.

3.1 Présentation du projet

Le projet est constitué de 188 spécifications rédigées en anglais, constituées d'une seule phrase, le prototype ne faisant pas le lien entre différentes phrases. Elles sont toutes classées en fonction de la difficulté du texte utilisé pour estimer la faisabilité de la traduction par le prototype. En tout, 33 sont considérées comme suffisamment basiques, avec une structure grammaticale simple et des compléments facilement identifiables pour pouvoir être traitées entièrement automatiquement sans erreur (exemple : « *For a specific zone of type DEFAULT, the area length shall be greater than MIN_SPECIFIC_ZONE_LENGTH* » [3]. Ce projet a déjà été traité manuellement, donc les résultats de ce test peuvent être comparés avec ce que les ingénieurs ont rédigé.

3.2 Analyse des résultats

3.2.1 Analyse des outils d'extraction d'informations

L'outil d'extraction d'entités, obtient des résultats très corrects. Près de 90% des entités voulues sont extraites, avec un peu plus de 10% de faux négatifs. De plus, seulement 2% des entités extraites sont des faux-positifs, ce qui fait un score F1 de 0,93.

L'étiquetage de relations n'est pas fiable. Les pourcentages de certitude de l'outil présenté ne dépassent pas les 20%, ce qui est trop faible pour différencier les vrais-positifs des faux-positifs par

2. clearsy.com
3. Les noms des données et le détail de la spécification ont été modifiées pour ne pas divulguer les informations confidentielles

un programme. Ces derniers sont donc très nombreux, par exemple dans la spécification « *The area length shall be greater than train length.* » , la relation correcte entre *area* et *length* de classe-attribut est trouvée avec une certitude de 16,4%, mais une relation fausse de même catégorie est aussi extraite entre *train* et *area* avec 13,44%, et il est difficile de trouver un seuil d'acceptation correct entre ces valeurs. Les explications possibles de ce manque de précision sont multiples, mais les principales pistes reposent sur le manque de données d'entraînement, ou un manque à l'entraînement d'entités n'ayant pas de relations entre elles. Pour compenser cela, une piste envisagée, autre que créer plus de données annotées, serait d'ajouter des informations connues du projet, sous la forme par exemple de graphes de connaissances.

La construction de l'arbre syntaxique diverge très rapidement des règles préétablies, avec des mauvaises liaisons trouvées entre les parties de la phrase, par exemple un intervalle avec les mot-clés *from* et *to* rattachés au verbe plutôt qu'au complément. Dès que les phrases emploient une structure un peu plus complexe que les schémas attendus, ces liaisons ne sont pas construites comme elles le devraient, et il faudrait à chaque fois rajouter une règle pour les traiter.

3.2.2 Analyse de la traduction

Sur les 188 spécifications :

- 6 ont été entièrement traduites, avec la bonne vérifications et les bons paramètres (3%),
- 154 sont partiellement traitées, avec une partie des paramètres ou la vérification non-reconnus (82%),
- 28 ne sont pas gérées par le prototype (15%), car reposant sur une structure grammaticale trop complexe.

Les règles entièrement traduites sont celles dérivées de l'exemple choisi comme base, présenté dans la figure 1. Seules les données changent, les formulations étant quasiment identiques.

En revanche, on s'aperçoit que le reste des spécifications sont assez mal gérées. Certaines plutôt simples grammaticalement sont quasiment entièrement traduites, mais pour la plupart il n'y a que certains morceaux qui sont reconnus, l'essentiel de la phrase restant trop complexe pour un prototype de cette portée. Tout d'abord, toutes les phrases utilisant d'autres formes syntaxiques que *sujet - groupe verbal - complément* ne sont pas traitées correctement, et pour toutes les spécifications qui s'étalent sur plusieurs phrases, il faut gérer les connexions entre celles-ci. Ensuite, même pour les phrases simples, le moindre changement dans les formulations utilisées peut faire basculer l'analyse de l'arbre syntaxique, et s'adapter à chaque situation s'avère pénible pour peu de gains.

L'analyse syntaxique montre dans cet exemple ses limitations, à savoir la rigidité en cas de légère variation. Ce prototype est peu utilisable dans un contexte industriel, mais les différents outils utilisés forment une base à améliorer pour extraire des informations plus précises.

4 Conclusion

La validation de données est une étape nécessaire pour le développement de systèmes critiques. Le principal travail des ingénieurs dans cette tâche est de traduire en règles formelles les spécifications écrites en langage naturel. Pour aider et accélérer ce processus, les outils de traitement de la langue semblent tout indiqués.

Plusieurs outils sont combinés afin d'extraire le plus d'informations possibles dans un prototype. Celui-ci associe ces informations avec des schémas existants afin de traduire automatiquement les spécifications les plus simples. Les résultats de l'utilisation de ce prototype sur un projet industriel montrent les limitations de cette approche, à savoir le manque de souplesse par rapport aux variations syntaxiques. Pour accomplir cette tâche de traduction, d'autres pistes sont envisagées, notamment du côté de l'analyse sémantique des phrases, aidées des connaissances implicites du projet via par exemple un graphe de connaissances.

Remerciements

Merci à Maximilien Colange, Catherine Dubois et Patrick Paroubek pour leur aide sur ce travail et la relecture de cet article.

Références

BROWN T. B., MANN B., RYDER N., SUBBIAH M., KAPLAN J., DHARIWAL P., NEELAKANTAN A., SHYAM P., SASTRY G., ASKELL A., AGARWAL S., HERBERT-VOSS A., KRUEGER G., HENIGHAN T., CHILD R., RAMESH A., ZIEGLER D. M., WU J., WINTER C., HESSE C., CHEN M., SIGLER E., LITWIN M., GRAY S., CHESS B., CLARK J., BERNER C., MCCANDLISH S., RADFORD A., SUTSKEVER I. & AMODEI D. (2020). Language models are few-shot learners.

DEVLIN J., CHANG M.-W., LEE K. & TOUTANOVA K. (2018). Bert : Pre-training of deep bidirectional transformers for language understanding.

HAN X., GAO T., YAO Y., YE D., LIU Z. & SUN M. (2019). OpenNRE : An open and extensible toolkit for neural relation extraction. In *Proceedings of EMNLP-IJCNLP : System Demonstrations*, p. 169–174.

HONNIBAL M. & MONTANI I. (2017). spaCy 2 : Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

LECOMTE T. & MOTTIN E. (2016). Formal data validation in the railways.

PAPANIKOLAOU Y., ROBERTS I. & PIERLEONI A. (2019). Deep bidirectional transformers for relation extraction without supervision.

SADOUN D. (2014). *Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot*. Theses, Université Paris Sud - Paris XI.

SHI P. & LIN J. (2019). Simple BERT models for relation extraction and semantic role labeling.