Extraction de contraintes dans des spécifications de validation de données

Arthur Remaud*

*Clearsy, Samovar, LISN,
arthur.remaud@clearsy.com

Résumé. Les spécifications de la validation de données, dans le domaine ferroviaire par exemple, sont majoritairement constituées de phrases dont les groupes verbaux expriment des contraintes à vérifier. Dans une optique d'automatisation du traitement de ces spécifications écrites en langage naturel, il est nécessaire d'identifier ces contraintes en utilisant des outils de traitement automatique de la langue. Nous avons expérimenté une approche utilisant un modèle BERT avec un apprentissage spécialisé. Une liste de contraintes à extraire, ainsi qu'un corpus de phrases et propositions syntaxiques a été élaboré pour l'entraînement, et un générateur de paraphrases a été utilisé pour pallier le manque de données. Les résultats sont encourageant, mais peuvent encore être améliorés, notamment en complétant les exemples pour l'entraînement.

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 et 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 leur taille importante, 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 formelle de données de l'entreprise Clearsy consiste en grande partie à traduire en règles formelles, les spécifications des clients concernant les propriétés à vérifier sur les données. Ces spécifications, rédigées en anglais, sont pour la plupart des phrases simples, que l'on voudrait traiter automatiquement ou semi-automatiquement, afin que les ingénieurs consacrent plus de temps aux spécifications plus complexes qui demandent plus de réflexion. Il est donc nécessaire d'extraire les informations contenues dans les spécifications, puis de raisonner dessus avant de générer la règle formelle finale.

Dans cette optique, une étape primordiale consiste à extraire les vérifications à effectuer. Par la suite, nous utiliserons le terme de contrainte pour parler d'une vérification à effectuer, car il s'agit bien de contraindre ces données à une règle établie. Des travaux ont déjà été réalisés de manière à extraire d'un texte, les phrases contenant des contraintes comme par exemple (Kiziltan et al., 2016), mais pas de les identifier. On peut également citer d'autres travaux qui utilisent des méthodes syntaxiques, comme (Anwar et al., 2020).

Dans une grande part des spécifications, la contrainte est exprimée par le ou les groupes verbaux. Dans cet article, nous ne parlerons donc que de l'extraction de contraintes basées sur le groupe verbal. Les contraintes exprimées dans les compléments sans verbe sont donc écartées, car il est plus difficile de bien les définir sans ambiguïtés, et cela augmenterait drastiquement le nombre de types de contraintes possibles. Par exemple, dans la phrase "*The name of a signal of type BUFFER shall begin by the prefix "BUF".*", on peut voir une contrainte sur un attribut (le type de signal traité), mais il ne sera retenu que la contrainte de début de liste (ici, une chaîne de caractères) exprimée par le groupe verbal *"shall begin by"*. Les spécifications sont découpées en propositions par la bibliothèque Python d'analyse de texte Spacy (Honnibal et Montani, 2017). Ces propositions doivent ensuite être analysées pour en extraire la contrainte.

Les méthodes syntaxiques pour identifier la contrainte d'un groupe verbal s'appuyant sur de la reconnaissance de patrons prédéterminés sont trop rigides, et ne s'adaptent pas à la diversité du langage (Remaud, 2021). Il faudrait en effet une quantité considérable de patrons pour pouvoir couvrir tous les cas possibles, ce qui demande beaucoup de ressources.

Nous proposons une approche par réseau de neurones utilisant un modèle BERT avec un apprentissage spécialisé pour pouvoir identifier la contrainte principale d'une proposition. A notre connaissance, il n'existe cependant aucun jeu de données de ce type en libre accès permettant un entraînement du réseau. Une liste de contraintes identifiables et un corpus d'exemples annotés ont donc été construit pour ce travail.

Dans cet article, nous décrivons une expérience mise en place pour pouvoir extraire et identifier les contraintes exprimées par des propositions de spécifications, dont le processus est décrit dans la Figure 1. Dans la section 2, nous montrons le choix de la liste des contraintes à extraire. Puis dans la section 3 nous détaillons la méthode utilisée pour identifier ces contraintes et la construction du corpus d'entraînement nécessaire pour la phase d'apprentissage de cette méthode avec les résultats que cela procure. Enfin dans la section 4 nous montrons l'utilisation d'un générateur de paraphrases pour augmenter les données, améliorant les résultats du modèle.

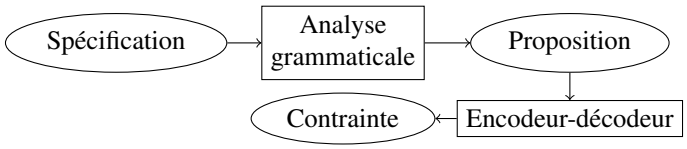

FIG. 1 – *Processus de recherche des contraitnes*

2 Construction de la liste des contraintes à extraire

Une liste de contraintes utilisées pour la programmation par contraintes a été élaborée dans le *Global Constraint Catalog* (N. Beldiceanu, 2010). Ce catalogue donne en particulier toutes les contraintes mathématiques, ensemblistes et logiques utilisées dans les problèmes de

satisfaction de contraintes. Nous l'utilisons pour établir notre liste de contraintes en ne retenant que celles les plus présentes dans les spécifications de validation de données.

Les complémentaires (ou opposés) de ces contraintes ont aussi été ajoutées. En effet, celles-ci peuvent prendre plusieurs formes, sans recourir à la négation syntaxique (par exemple, *"is different from"* est l'opposé de *"is equal to"*).

A cette liste préétablie ont été rajoutées des contraintes rencontrées dans des spécifications des projets de l'entreprise Clearsy. Celles-ci portent notamment sur la structure de données, comme l'unicité d'un élément dans une liste (ex : *"An identifier shall be unique in the identifier list of all components"*).

Au total, 23 contraintes à extraire ont été retenues pour notre expérimentation. Ce nombre n'est pas conditionné par les outils, et peut être augmenté dans une utilisation future pour inclure plus de contraintes dans les recherches.

3 Choix du modèle et construction d'un corpus pour l'apprentissage

3.1 Approche par réseau de neurones

Il existe plusieurs méthodes pour l'extraction d'informations dans du texte. Dans un précédent travail (Remaud, 2021), il a été constaté que l'identification de contraintes par règles syntaxiques suivant des patrons prédéterminés est trop rigide, et que le moindre éloignement de ces patrons empêche la reconnaissance.

Des travaux similaires (Kiziltan et al., 2016; Winter et Rinderle-Ma, 2018) utilisent des approches de traitement de la langue qui donnent de bonnes performances en utilisant des méthodes statistiques basés sur des sacs de mots. Ces méthodes ne suivent plus l'état de l'art, plus tourné sur l'apprentissage profond. De plus, elles reposent généralement sur de l'apprentissage sur de grand corpus de textes, que nous ne possédons pas.

À l'écriture de cet article, l'approche la plus performante dans le cas général d'extraction d'informations est l'utilisation d'un réseau de neurones encodeur-décodeur. Ce réseau est pré-entraîné sur un nombre important de textes pour obtenir des plongements lexicaux du texte, représentant sa sémantique sous forme de vecteurs numériques de grande dimension. Ensuite, à l'aide d'un apprentissage de spécialisation, ce réseau de neurones peut apprendre une tâche en particulier sur cette représentation. Cette spécialisation a besoin d'un corpus moins important que le pré-entraînement, ayant déjà acquis des informations sémantiques lors du pré-entraînement.

L'avantage de l'approche neuronale est l'abstraction de la grammaire par l'utilisation des plongements lexicaux. Ainsi toutes les notions de sémantiques sont contenus dans ces plongements, et ce sans prévoir à l'avance les différentes représentations possibles dans le texte.

Dans notre cas, la tâche consiste en l'identification de la contrainte dans une proposition.

Il a donc été choisi d'utiliser ici cette approche. L'un des modèles d'encodeurs-décodeurs existants le plus courant est le modèle BERT (Devlin et al., 2018), dont l'apprentissage des plongements lexicaux a déjà été opéré et dont il ne reste plus que l'apprentissage spécialisé à faire pour le travail voulu. Ce modèle existe sous de nombreuses déclinaisons afin d'être spécialisé dans de nombreuses tâches.

Le modèle précis sur lequel se base cette expérience est le modèle **BertForSequenceClassification**, disponible dans la bibliothèque de réseaux de neurones huggingFace (Wolf, 2019). L'interface de ce réseau de neurones est normalisé ce qui facilite son utilisation. Il est conçu spécifiquement pour la classification de texte, et nous l'avons adapté pour pouvoir ranger des phrases parmi 25 classes différentes, correspondant à nos 23 différentes contraintes plus deux cas particuliers :

— la proposition contient une contrainte mais il n'est pas possible de l'identifier, ou elle est absente de la liste des contraintes à détecter (ex : "*The certificate must be validated as appropriate for the security requirements*" [1], où la contrainte est méthodologique et n'est donc pas utile pour une automatisation),
— la proposition ne contient pas de contraintes (ex : "*The Density of the CSP network depends on the number of meetings (m).*" [2]).

Notre approche nécessite un corpus d'entraînement pour cette phase d'apprentissage spécialisé, qui a été conçu pour cette expérience. Celui-ci est composé de phrases et de propositions étiquetées avec l'une des classes de la liste décrite précédemment.

3.2 Ressources utilisées pour le corpus

Les premiers exemples pour l'entraînement du réseau de neurones ont été pris dans les spécifications de projets de validation de données de l'entreprise Clearsy. Ils fournissent déjà une part considérable d'exemples, mais utilisent des formulations très similaires, ce qui est courant et recommandé au sein d'un même projet pour rester cohérent. Il faut donc aussi des exemples venant d'autres sources permettant de varier l'apprentissage.

Pour compléter les données avec des exemples plus neutres et ne venant pas de projets de validation de données, d'autres sources publiques ont été exploitées. Ces recherches ont été faites manuellement en extrayant les phrases utilisant des verbes modaux d'obligation comme "*shall*", "*must*", "*should*" ou "*can*".

La première source d'exemples est le *Global Constraint Catalog*, cité précédemment, qui contient des phrases de démonstration ou d'explication par contraintes, qui sont réutilisées ici.

Puis des problèmes de programmation par contraintes ont été ajoutés, venant de la base de données CSPlib (csp, 1999). Cette bibliothèque de problèmes pour évaluer les outils de programmation par contraintes recense 91 problèmes auxquels une grande partie des énoncés est assimilable à des contraintes que l'on retrouve dans la validation de données.

La troisième source sont les *Request for comments* (RFC), documents en langue anglaise décrivant des caractéristiques et spécifications techniques et disponibles à tous en ligne afin d'arriver à un consensus dans la communauté concernée (Butler, 2016; Bray, 2014; Cooper, 2008; Ash, 2005; T. Berners-Lee, 1995; Fielding, 1999; S. Turner, 2010; Shelby, 2014). Cela est notamment très utilisé pour les protocoles d'internet afin que tous les ordinateurs utilisent les mêmes principes pour communiquer. La liste de RFC pouvant être inclues aux données d'entraînement peut encore être étendue, mais a été reportée par manque de temps.

1. Exemple pris dans la RFC 7252
2. Exemple pris dans le problème 46 de CSPLib

Ensuite, des documents de normes officielles tels que les normes UML (UML, 2009), OCL (UML, 2006), DOL (DOL, 2018), OWL (OWL, 2012) ou XML (XML, 2008) ont été explorés pour trouver des contraintes. Là aussi la liste n'est pas exhaustive, et d'autres normes peuvent être ajoutées pour compléter les données.

Pour finir, des phrases improvisées par les auteurs ont été ajoutées, pour compléter principalement les contraintes avec le moins d'exemples associés et donc moins bien identifiées. Ces phrases essayent de s'éloigner le plus possible des exemples déjà existants afin d'éviter toute redondance qui limiterait l'apprentissage.

Avec l'ensemble de ces documents, nous arrivons à un total de 1068 exemples. Près de la moitié de ceux-ci sont issus des projets de validation de données. La répartition des sources parmi les exemples est donné dans la Figure 2. Une partie de ces phrases est conservée pour les tests et ne sera pas utilisée pour l'apprentissage spécialisé. Cela donne 961 propositions pour l'entraînement et 107 pour le test en suivant un ratio de répartition de $0,9$.

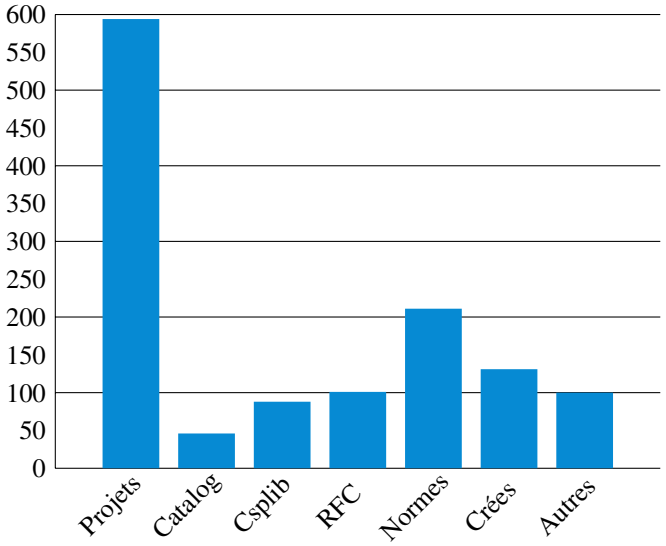

FIG. 2 – *Répartition des sources dans les données d'entraînement*

3.3 Entraînement spécialisé et analyse des résultats

Un premier entraînement a été fait avec ces données restreintes. La moyenne des scores F1 de ce modèle pour chaque classe est de seulement 52%. La faiblesse des résultats peut principalement s'expliquer par le manque de données, ce qui empêche le modèle de converger correctement. Pour pallier à ce problème, une piste est d'utiliser de l'augmentation des données.

4 Augmentation des données par avec des paraphrases

4.1 Le générateur de paraphrase

Face au manque de données différentes dans les projets de l'entreprise, et à la quantité de main-d'œuvre importante à déployer pour rechercher d'autres contraintes dans d'autres sources externes ou pour produire de nouvelles contraintes originales, il a été décidé de faire de l'augmentation de données. En traitement de la langue, il est possible d'utiliser un outil de construction de paraphrases, et le choix s'est porté sur Parrot (Damodaran, 2021). A partir d'une phrase donnée en entrée, il génère plusieurs autres phrases similaires en remplaçant des mots par des synonymes, ou modifiant légèrement les formulations syntaxiques utilisées pour obtenir des variations de l'exemple de base. A chaque fois, la contrainte annotée à la phrase d'origine a été conservée pour les paraphrases générées.

Grâce à cet outil, les données pour l'entraînement passent de 1068 à 3132 exemples, soit près du triple. Même si ces phrases ajoutées sont très similaires à celles de base, cela permet au réseau de neurone d'apprendre des nouvelles formes de texte, différentes des formulations présente dans le corpus.

Finalement, notre corpus compte 2818 exemples pour l'entraînement, avec 314 pour les tests en respectant le ratio initial de $0,9$.

4.2 Résultat du modèle avec ces nouvelles données

Le même entraînement a été effectué sur ces données augmentées, avec les mêmes paramètres qu'auparavant (système de batch identique, même nombre d'epochs). Bien entendu le temps d'apprentissage est beaucoup plus long dans cette configuration.

L'ajout des paraphrases a un fort impact sur les résultats de moyenne de score F1, qui passent à 87%. Cela montre l'importance de ce type d'outil pour entraîner un réseau de neurones lorsque l'on manque d'exemples annotés.

Les données utilisées pour l'entraînement peuvent encore être complétées, à partir de plus de projets de validation de données, ou en ajoutant de nouvelles sources de documents. Les résultats présentés dans cet article peuvent donc être améliorés, notamment pour reconnaître l'absence de contraintes dans une phrase qui peut prendre beaucoup de formes différentes.

5 Conclusion

L'identification des contraintes exprimées dans une phrase est nécessaire pour l'automatisation du traitement de spécifications de validation de données. Pour cela, une approche utilisant un réseau de neurones sur des propositions syntaxiques a été proposée.

Une liste de 23 contraintes à identifier a été établie en se basant sur le *Global Constraint Catalog* (N. Beldiceanu, 2010) et les projets de validation de données. Cette liste pourrait être amenée à être complétée à l'avenir avec de nouvelles contraintes rencontrées fréquemment. En plus de cette liste, sont identifiés les cas où une contrainte est trouvée mais non reconnue, et les cas où aucune contrainte n'est extraite.

Un corpus a été élaboré à partir de projets internes de validation de données, et de ressources disponibles librement comme de la documentation ou des normes. Le nombre d'exemples reste cependant restreint à cause du manque de main-d'œuvre pour construire ce corpus. Afin d'améliorer les performances, un outil de génération de paraphrases, nommé Parrot (Damodaran, 2021), a été utilisé afin de multiplier par 3 les données d'entraînement, et ainsi mieux traiter les modifications légères du texte.

Deux apprentissages ont été expérimentés : un avec les données construites manuellement et un second avec en plus pour l'entraînement les paraphrases créées par le générateur automatique. Pour le premier, les résultats ont une précision de 52%, alors que le deuxième présente des résultats atteignant 87%. Cela montre l'importance de l'augmentation de données pour l'apprentissage de réseaux neuronaux sur des tâches de traitement de la langue.

Les résultats actuels sont prometteurs, mais ils peuvent être améliorés en augmentant la base d'exemples pour l'apprentissage. Cela se voit notamment pour les contraintes qui possèdent peu de données à l'entraînement.

Après les contraintes du groupe verbal, nous nous intéresserons à l'analyse des sujets et compléments constituant les paramètres de la contrainte, avec les données du projet de validation qu'ils contiennent mais aussi les restrictions sur ces données qui ne se reflètent pas dans le groupe verbal, et qui ont de l'importance sur la vérification finale à appliquer.

Remerciements

Un grand remerciement à Catherine Dubois, Maximilien Colange et Patrick Paroubek pour leur relecture et correction de cet article. Les auteurs tiennent aussi à remercier l'ANRT pour sa collaboration à la thèse CIFRE n°2020/0392.

Références

(1999). CSPLib : A problem library for constraints. `http://www.csplib.org`.

(2006). Object constraint language. Technical report, Object Management Group.

(2008). Extensible markup language (xml) 1.0 (fifth edition). Technical report, W3C.

(2009). Omg unified modeling language (omg uml). Technical report, Object Management Group.

(2012). Owl 2 web ontology language structural specification and functional-style syntax (second edition). Technical report, W3C.

(2018). The distributed ontology, modeling, and specification language. Technical report, Object Management Group.

Anwar, M., I. Ahsan, F. Azam, W. Haider, et M. Rashid (2020). A natural language processing (nlp) framework for embedded systems to automatically extract verification aspects from textual design requirements. pp. 7–12.

Ash, J. (2005). Max Allocation with Reservation Bandwidth Constraints Model for Diffserv-aware MPLS Traffic Engineering & Performance Comparisons. RFC 4126, RFC Editor.

Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor.

Butler, e. a. (2016). The GeoJSON Format. RFC 7946, RFC Editor.

Cooper, e. a. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, RFC Editor.

Damodaran, P. (2021). Parrot : Paraphrase generation for nlu.

Devlin, J., M.-W. Chang, K. Lee, et K. Toutanova (2018). Bert : Pre-training of deep bidirectional transformers for language understanding.

Fielding, e. a. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor.

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

Kiziltan, Z., M. Lippi, et P. Torroni (2016). Constraint detection in natural language problem descriptions. In *IJCAI*, pp. 744–750.

Lecomte, T. et E. Mottin (2016). Formal data validation in the railways.

N. Beldiceanu, M. Carlsson, e. J.-X. R. (2010). Global Constraint Catalog, 2nd Edition. https ://sofdem.github.io/gccat/gccat/titlepage.html.

Remaud, A. (2021). Utilisation d'outils de tal pour la compréhension des spécifications de validation de données. *HAL*.

S. Turner, S. C. (2010). Clearance Attribute and Authority Clearance Constraints Certificate Extension. RFC 5913, RFC Editor.

Shelby, e. a. (2014). The Constrained Application Protocol (CoAP). RFC 7252, RFC Editor.

T. Berners-Lee, D. C. (1995). Hypertext Markup Language - 2.0. RFC 1866, RFC Editor.

Winter, K. et S. Rinderle-Ma (2018). *Detecting Constraints and Their Relations from Regulatory Documents Using NLP Techniques*, pp. 261–278.

Wolf, e. a. (2019). Huggingface's transformers : State-of-the-art natural language processing. *ArXiv abs/1910.03771*.

Summary

Data Validation specifications are mainly made of sentences where verbal groups give some constraints to be verified. For the purpose of an automated treatment of these natural language specifications, we need to extract and identify those constraints using natural language processing tools. We present in this article an experimentation with a neural network model based on BERT fine-tuning. A list of constraints to identify and a corpus of sentences and syntactic propositions have been created, and a paraphrase generator has been used to balance the lack of training data. Results are promising but can be nevertheless improved, for example by increasing the quantity of data.