CNL 2020/21

Seventh International Workshop on Controlled Natural Language

Workshop Proceedings

8 - 9 September 2021

Preface

CNL 2020/21 is the seventh edition of the workshop series on Controlled Natural Language (CNL)[1]. It was initially planned for 2020, but had to be postponed by one year due to the Covid-19 pandemic. It is co-located with the SEMANTiCS 2021 conference[2] and will be held on 8 and 9 September 2021 in Amsterdam, as a hybrid event where onsite as well as online participation is possible.

We received 15 full paper submissions and two submissions as short papers. These 17 papers then received in total 54 reviews from the members of the program committee, which corresponds to an average of 3.2 reviews per paper. Out of the full paper submissions, eleven were accepted as such (acceptance rate of 73%). Of the remaining four, three were rejected as full papers but accepted as short ones. The two short paper submissions were both accepted. Therefore, these proceedings include eleven full papers and five short ones.

We would like to thanks all authors for their submissions and the program committee members for their careful reviews. We are now looking forward to the workshop with this exciting program.

Tobias Kuhn
Silvie Spreeuwenberg
Stijn Hoppenbrouwers
Norbert E. Fuchs

[1] http://www.sigcnl.org/cnl2020.html
[2] https://2021-eu.semantics.cc/

Organization Committee

- Tobias Kuhn (VU Amsterdam, Netherlands)
- Silvie Spreeuwenberg (LibRT, Netherlands)
- Stijn Hoppenbrouwers (HAN University of Applied Sciences and Radboud University, Netherlands)
- Norbert E. Fuchs (University of Zurich, Switzerland)

Program Committee

- Krasimir Angelov (Digital Grammars, Sweden)
- Mihael Arcan (National University of Ireland, Galway)
- John Camilleri (Digital Grammars, Sweden)
- Brian Davis (Maynooth, Co Kildare, Ireland)
- Ronald Denaux, (Expert System, Spain)
- Ramona Enache (Microsoft, Sweden)
- Sebastien Ferre (University Rennes 1, France)
- Antske Fokkens (VU Amsterdam, Netherlands)
- Albert Gatt (University of Malta)
- Normunds Gruzitis (University of Latvia)
- Yannis Haralambous (IMT Atlantique, France)
- Herbert Lange (University of Gothenburg, Sweden)
- Kaarel Kaljurand (Nuance Communications, Austria)
- Maria Keet (University of Cape Town, South Africa)
- John P. McCrae (National University of Ireland, Galway)
- Roser Morante (VU Amsterdam, Netherlands)
- Gordon Pace (University of Malta)
- Laurette Pretorius (University of South Africa, South Africa)
- Rolf Schwitter (Macquarie University, Australia)
- Giovanni Sileno (University of Amsterdam, Netherlands)
- Irina Temnikova (Qatar Computing Research Institute, Qatar)
- Mike Rosner (University of Malta)
- Camilo Thorne (Elsevier, Germany)
- Adam Wyner (Swansea University, UK)

Table of Contents

Towards a More Natural Controlled Language in Future Airbus Cockpits. A Psycho-linguistic Evaluation

Nataly Jahchan[1], Anne Condamines[2], Emmanuelle Cannesson[3], and Hélène Giraudo[4]

[1,3]Airbus Operation SAS
[1,2,4]CLLE-CNRS
{nataly.jahchan,emmanuelle.cannesson}@airbus.com
{anne.condamines,helene.giraudo}@univ-tlse2.fr

Abstract

The main goal of this research is to optimize an existing Airbus Cockpit Controlled Language in order to integrate it in future cockpit design. The current controlled language used aboard Airbus cockpit interfaces was carefully constructed to avoid ambiguity and complexity. In order to optimize the existing language, we set out to evaluate the appropriate levels of simplification that would achieve more accurate and faster comprehension with optimized pilot training time by using psycho-linguistic experimentation and cognitive science tools. We present in this paper a congruency task similar to traditional judgment tasks in behavioral experiments. It provides a firmly controlled environment to test linguistic hypotheses and CNL rules. Results show that what we sometimes mistakenly label as superfluous or empty syntactical elements could go a long way in ensuring better comprehension and faster information processing from a psycho-linguistic point of view.

1 Introduction

The main goal of this research is to optimize an existing Airbus Cockpit Controlled Language in order to integrate it in future cockpit design. The current controlled language used aboard Airbus cockpit interfaces was carefully constructed to avoid ambiguity and complexity (as are all comprehension oriented controlled languages, (Kuhn, 2014; Schwitter, 2010; Kitteridge, 2003) and is designed to help pilots operate and navigate the aircraft (with the help of cockpit screen interfaces) in normal and abnormal (in cases of emergency or failures) situations. The need for clear and unambiguous communication is vital in safety critical domains. This controlled language and the rules that make it were put in place at a time when design flexibility was limited (for example small screen sizes that restrict word and sentence length (Spaggiari et al., 2003; Jahchan et al., 2016; Jahchan, 2017). This results in a CNL which is non-conforming to natural language syntax, highly abbreviated, typographically variable, and color-coded (Jahchan, 2019). As we are addressing a more flexible disruptive cockpit design for future aircraft, these limitations are no longer immutable constraints, and the future controlled language need not be so coded and compact, or follow very strict simplification rules.

The goal being to take into consideration the disruptive cockpit design (possibly larger screen sizes (less character limitations), newer technology, etc.) which goes hand in hand with an adapted human-oriented controlled language and which is safe, suitable and easily accessible for a human operator.

Therefore, in order to optimize the existing language, we set out to evaluate the appropriate levels of simplification that would achieve more accurate and faster comprehension with optimized pilot training time by using psycho-linguistic experimentation and cognitive science tools. In order to determine the appropriate levels of simplification, one must carefully investigate the problem in context (operational piloting constraints, cockpit design constraints, linguistic ambiguities (syntactic, semantic, and terminological ones). In this sense, we are more particularly dealing with Ergonomic Linguistics (Condamines, 2021) in which linguistic models, theories, and hypotheses are used in specified work contexts (mainly in industry) to achieve precise goals efficiently and serve a real life

operational purpose (one of the primary uses of Human-oriented CNLs). These hypotheses and propositions are derived from real language productions and theoretical linguistic theories (for example common CNL construction rules among several languages (O'Brien's, 2003) and should be evaluated using experimental techniques and acceptability tests to acquire empirical evidence to support their efficiency when it comes to comprehension and optimal performance for human operators (target users of CNLs). This concept is closely related to readability and usability. Our own definition of readability for the purposes of this research does not involve the traditional definition, i.e. ease of reading, reading proficiency, or the characteristics that make readers willing to carry on reading (Flesch Kincaid, Smog formula, (Flesch 1979), etc.). Readability in our sense is about usability of the text. Usability is defined as the *"extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"* (ISO/DIS 9241-11.2 :2016).

To this date, CNL evaluations are not systematically enforced and very rarely put in place for human-oriented CNLs. There have been some evaluations of CNLs using NLP (natural language processing) tools in corpus linguistics-based approaches such as the verification of conformity of requirements (Condamines and Warnier, 2014; Warnier, 2018) or for text complexity (Tanguy and Tulechki, 2009), and machine translation (O'Brien and Roturier, 2007; Aikawa et al., 2007), or for syntactic transformations and corpus alignment of specialized corpora with existing simplified corpora (Cardon and Grabar 2018), etc. There have also been evaluations based on ontographs for knowledge representation and formal languages (Kuhn, 2010). In this paper, Kuhn (2010) contends that *"user studies are the only way to verify whether CNLs are indeed easier to understand than other languages"*. He argues that it is difficult to obtain reliable approaches with task-based and paraphrase-based evaluation approaches, and offers an alternative method for evaluating formal logic-based languages. Consequently, existing CNL research falls short on providing empirical proof on the effectiveness of comprehension-oriented CNLs on the human cognitive processes of language comprehension, for instance by measuring reaction times and accuracy in performance. We argue that the relative lack of cognitive behavioral evaluations is equivalent to rendering CNLs mere style guides or good authoring practices, and the reasons for adopting certain rules over others are unreliable.

Uncontrolled natural language is ambiguous and unsuitable for use in domains where ambiguity may be dangerous such as the aviation industry, but on the other hand, it represents an intricate part of our cognitive processes and its rules must not be excluded. Readability, text simplification, and text complexity research have focused on simplifying the language by making it less and less like natural language, and more like an unambiguous set of codes and regulations so that the resulting language veered away from the "natural" dimension. But to what extent is text simplification satisfactory and what are the limits at which it becomes counter-productive? When must natural language structures be respected? We constructed a more natural controlled language (MNL) by basing ourselves on the existing more codified controlled language (MCL) and its operational needs, syntactic and terminological rules) by using research that has been done on readability and text complexity and test, bit by bit, how we can add sentential elements that would make the language closer to natural language structure of English. At the same time, by adding a sentence structure we would be limiting the different possible interpretations, therefore avoiding, as much as possible, elliptical ambiguities (C.f Figure 1)

Figure 1, Example of MCL and MNL

Although pilots are trained to understand the meaning of the typographical ellipses (dots separating "engine" and "off" and color coding to mean an action that must be performed, the sentence structure (in the proposed more natural format) provides a fail-safe way of avoiding ambiguity. The sentence "Turn off the engine" adds two more words to the original statement "engine.....off " yet completely eliminates the second possible interpretation (the engine is off). Thus, information is solely contained

in the linguistic elements, excluding color and typographical separation. There is only one possible way of interpreting and understanding the second sentence. In this way, we based ourselves on the MCL corpus (operational use and context, goal) and created new more natural structures (MNL) to be evaluated.

2 Method

As a first approach, we used congruency tasks to evaluate passive comprehension. To be able to use congruency tasks (commonly used in cognitive psychology experimentation) we had to limit ourselves to the use of the "information category" in our corpus, and more particularly, the constative messages informing pilots of the availability of a certain function such as "Galleys extraction available in Flight" or "Expect high cabin rate". These sentences do not require direct action but comprehension and awareness on the pilot's end (c.f Figure 2).

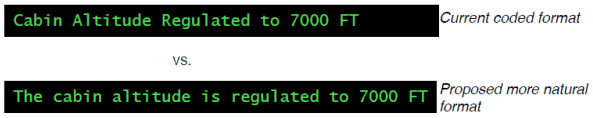


Figure 2, Example of MCL and MNL in an Informational Statement

2.1 Construction of Messages

In the following example case, the original coded and abbreviated message is L TK 17000 KG MAX AVAIL which when decoded without abbreviations means "left tank 17000 kilograms maximum available". It was relatively easy to construct the MCL messages since we could keep the same structure and same words when possible, and find or construct an image that is congruent to its meaning. However, constructing the equivalent MNL messages was a little more complicated as we had several options; there was at least 4 different ways of writing the sentence in the previous example in a more natural language (cf. Jahchan, 2019).

1. There are maximum 20 kilos available in the left container
2. There are 20 kilos maximum available in the left container
3. The left container has maximum 20 kilos available
4. The left container has 20 kilos maximum available

After careful consideration and in order not to multiply variables, we chose the first option for the MNL structure as the existential clause "there is/are" introduced by the expletive pronoun "there" + predicate "are" indicates the existence or the presence of something in a particular place or time, which in our experiment reinforced the idea of something available or not available in the target picture. The existential clause itself expresses a predicate of existence which sets the tone for the incoming noun phrase. While the second option also includes an existential clause, it was not deemed sufficiently plausible by English native speakers that we consulted. The existential clause introduced in the MNL structures also inverts the theme and rheme structure of the original MCL structure. The current controlled language uses the theme at the onset of the message "left container" followed by the rheme. One of the main differences between both languages is the addition of function words in the MNL stimuli. Leroy et al. (2010) affirms in a study about the effects of linguistic features and evaluation perspectives that *"complex noun phrases significantly increased perceived difficulty, while using more function words significantly decreased perceived difficulty. [...] Laypersons judged sentences to be easier when they contained a higher proportion of function words. A high proportion of function words leads to a different cadence closer to spoken language. It may also help space out individual concepts in text to facilitate assimilation."*

2.2 Stimuli

We created a new corpus of messages inspired by everyday life situations to test our hypothesis with naïve participants that are not familiar with aeronautical corpus terms. An example of this sentences is "parking spot is available", that emulate the syntax and intentions of our original corpus statements. As a first step, the newly proposed structures were purposefully tested on naïve participants (and not pilots) to avoid expert bias and determine comprehension and performance levels on a more general level. The corpus was divided in 6 difficulty categories that represent syntactical structure of the information availability statements. They went from 1 the

easiest structure (noun + nous + available) to 6 most difficult (noun + noun + noun + available + in +noun) as length has been proven to be an effective and efficient index of syntactic difficulty Szmrecsanyi (2004). According to Szmrecsanyi (2004), sentence length (or a version of the Flesch-Kincaid tests) are as good a means of testing syntactic text complexity as counting syntactic nodes in a sentence. Szmrecsanyi reports comparing three methods of measuring syntactic complexity node counts, word counts, and 'Index of Syntactic Complexity' (which takes into consideration the number of nouns, verbs, subordinating conjunctions, and pronouns). She concludes that the three measures are near perfect proxies since they significantly correlate and can be used interchangeably. Once the messages were set, we looked for, constructed, or modified existing real life images which accurately portrayed the messages we previously concocted, which have similar syntactic structure and difficulty as messages present in the original corpus (MCL), and for which we created a corresponding MNL version (c.f. Figure 3)

Non-Aviation Messages Parallel to ECAM Structure Messages	Syntax (Difficulty 1-6)
Chalk board available	1- Noun + Noun + Avail
Mobile car holder available	2- Noun + Noun + Noun + Avail
Emergency exit available in building	3- Noun + Noun + Avail + In + Noun
Office writing supplies available in catalogue	4- Noun + Noun + Noun + Avail + In + Noun
Left container 20 kilos maximum available	5- Adj + Noun + Num + Noun + Noun + Avail
Yellow hall 2 movie posters minimum available	6- Adj + Noun + Num + Noun + Noun + Noun + Avail

Figure 3, Example of 6 conditions of difficulty

As messages were different in length, the allotted reading time was different depending on the number of words. MNL messages necessarily have more words than MCL messages. However, those words were only grammatical words such as "there is" or "a", or "the", etc. We decided to count only lexical words to calculate reading time. This choice might have inadvertently given a position of privilege to the MCL messages since MNL messages had more total words (grammatical and lexical) than the equivalent MCL messages yet they had the same reading time (same number of lexical words). We based ourselves on word per minute and reading time research to calculate the time the messages appeared on the screen (Trauzettel-Klosinski Dietz, 2012).

2.3 Experimental Design and Participant Task

Before beginning the experiment, participants filled out different forms: a general ethics and compliance consent form, a data sheet in which they specified their age, gender, dexterity, native language, English placement, knowledge of Airbus Control Language. All non-native English speakers also performed a quick English placement test online to determine their CEFR levels (Common European Framework of Reference for Languages). The levels range from A1 or breakthrough/ beginner to C2 or Mastery/Proficiency.

Participants started with a practice session composed of a different set of 24 semi-randomized stimuli representative of the difficulty and language conditions, and the same image construction methodology as the target stimuli in the main lists. They had noise cancelling headphones and were set in a quiet room with no distractions. Each list consisted of 48 target stimuli, split into 24 congruent stimuli (image congruent with the message, correct answer is a "yes") and 24 incongruent stimuli (image incongruent with the message, correct answer is a "no"). Participants had 5000 ms to respond. This time lapse was validated by doing several pretests to ascertain the adequate display time for reading the messages. In case of a non-answer the next stimulus appears and so on. Once the participant responds the image disappears and the next fixation cross appears. The task consisted of the participants reading a text written in either the More controlled Language (MCL) syntax or the More Natural Language (MNL) syntax (c.f. Figure 4)

The messages appear out of context preceded only by a 3000 ms fixation cross in the middle of the screen. We decreased that value to 150 words per minute (WPM), so that a message that has 3 lexical words would appear for 1.2 seconds (3 x 60/150) and a message that has 6 lexical words would appear for 2.8 seconds (6 x 60/150), etc. The text (the prime) then disappears and a target image appears, an image which could be congruent with the previously read text or incongruent. I.e. if the text says "bus stop available" and the image shows a bus stop then the participant has to press "yes" on the controller to indicate congruency, and if for instance the

image shows an image of a car then the participant should press on "no" to indicate that the image is incongruent with the text.

Response times and precision in both language conditions were recorded. We chose sentences that could show an accurate visual description of a situation or scene.

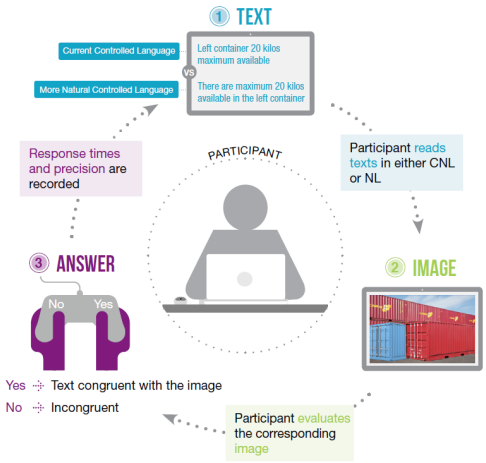


Figure 4, Representation of Task Performance

2.4 Participants

72 participants took part in the first experiment (12 native speakers of English and 60 non-native speakers whose placement levels ranged from A1 to C2 in CEFR). The non-native speakers' languages included Arabic, Chinese, Dutch, French, German, Portuguese, Spanish, Serbian, and Indonesian, with the overwhelming majority being French (45 out of 60). 38 participants had no knowledge whatsoever of controlled languages. 16 claimed had beginner knowledge of the Airbus controlled language (Airbus employees having rarely worked with the language or its rules). 14 had a more intermediate knowledge of the language. 5 participants had expert knowledge of the language as it could be part of their daily task.

2.5 Experimental Materials and Equipment

DMDX is a Win 32-based display system used in psychological laboratories to measure reaction times to visual and auditory stimuli. We used this software on a Dell Precision 3510 laptop to display the messages and images. For that, we developed 6 scripts which consisted of 3 semi-randomized lists of stimuli for right-handed participants and 3 for left-handed participants (same lists but the "yes" and "no" buttons were inverted for left handed participants).

2.6 Variables

The list of independent variables that we will evaluate are:

- Language (MCL-MNL)
- Syntactic Difficulty (1 to 6)
- Type (Congruents-Incongruents) Extraneous and participant variables:
- English placement level (Basic Intermediate, Proficient, Mastery, Native)
- Familiarity with Airbus CL (None, Beginner, Intermediate, Expert)

Dependent variables:

- Reaction time in ms, Accuracy (number of errors)

2.7 Hypotheses and Research Questions:

1. MNL messages produce shorter reaction times than MCL ones in different syntactic difficulty conditions.
2. MNL messages produce less errors (are more accurate) than MCL ones in different syntactic difficulty conditions.
3. Did the language factor play a different role for the different types of congruency responses regarding reaction times?
4. Did the language factor play a different role for different levels of English placement (Basic Intermediate, Mastery, Natives) regarding reaction times?

3 Results and Statistical Analysis

We reported the results below linked to each of the previously mentioned hypotheses. We used non-parametric statistical significance tests such as Wilcox signed rank as the data had a non-normal distribution (Gaussian distribution). These tests help determine whether the independent variables had an effect on reaction time and accuracy of comprehension (dependent variables) by calculating a statistical significance p-value (results are significant if they show a p-value less than 0.05, i.e. implying that it is acceptable to have less than 5% probability of incorrectly rejecting the true null hypothesis). **1. MNL messages produce shorter reaction times than MCL**

ones in different syntactic difficulty conditions. A Shapiro-wilk normality test was run on the reaction times and the results showed that the data is significantly non-normal (p = 2.054e-05) with abnormal skew, therefore we used non-parametric tests to test the main effect such as the Wilcox signed rank test because the same participants took part in both language conditions. Firstly, the general effect was compared regardless of difficulty for both language conditions. There was a significant difference in the scores for MCL (Median=2030.317 ms.) and MNL (Median= 1944.163 ms.) conditions; v=1692, p=0.0339, effect size calculated with Pearson's coefficient r=0.24998. With the hypothesis confirmed, we can conclude that the more natural language helped participants process the stimuli and provoked significantly faster reaction times than the more coded language format.

We then performed a linear regression model to ascertain the influence of the syntactic difficulty condition in both languages. A simple linear regression was calculated to predict the reaction times of the MCL responses based on the 6 syntactic difficulty conditions. A significant regression equation was found ($F(1,1500) = 9.211$, $p < 0.002447$), with an R2 of 0.006103. Participants' predicted reaction times is equal to 1873.77 + 42.55 ms for every additional difficulty condition. Therefore, reaction time increased 42.55 ms for each additional difficulty condition. A simple linear regression was also calculated to predict the reaction times of the MNL responses based on the 6 difficulty conditions. A significant regression equation was found ($F(1,1450) = 12.68$, $p < 0.0003822$), with an R2 of 0.008667. Participants' predicted reaction times is equal to 1801.64 + 47.81 ms for every additional difficulty condition. Therefore, reaction time increased 47.81 ms for each additional difficulty condition. Figure 5 is the graph that plots those two linear regression models for both languages in the 6 difficulty conditions. As we can see there is no interaction between the two languages (lines are parallel and do not intersect) but reaction times get slower when difficulty increases in both languages which confirms that syntactic difficulty based on length is a valid measure (confirms Szmrecsanyi (2004) findings). With the hypothesis confirmed, we can also conclude that MNL messages produced consistently faster reaction times than MCL messages in all difficulty conditions.

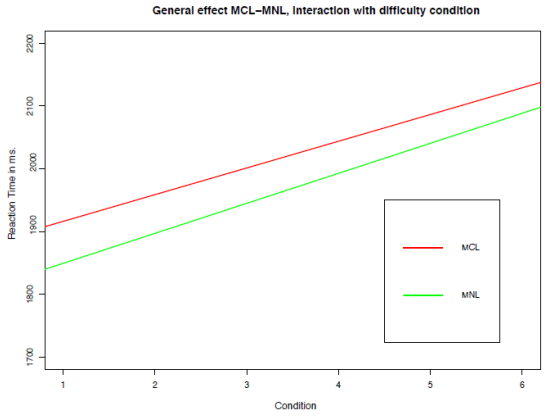


Figure 5, Linear Regression Models for MCL and MNL Difficulty Condition

2. MNL messages produce less errors (are more accurate) than MCL ones in different syntactic difficulty conditions. Accuracy was calculated using the average number of errors. Therefore, we started by comparing the general effect of accuracy regardless of difficulty for both language conditions using the Wilcox signed Rank test. There was no significant difference in the number of errors by subject produced in the MCL (Mean = 2.46 errors) and MNL (Mean = 2.9 errors) conditions; v = 549, p = 0.07121. We could interpret this by proposing that the difference in the syntax of the two languages was not different enough (a lot of the stimuli had only one or two grammatical articles added to them) to cause one language to have better performance with respect to errors, but those subtleties were manifested in the reaction times instead which stand to be more adequate measures of early/initial comprehension. Figure 6 is a histogram plot of the errors made in the different conditions of difficulty for both languages. As we can see the number of errors in both languages is not consistent across different difficulty conditions, but there is a tendency for both languages to have more and more mistakes as difficulty increases. The advance that the MCL has over the MNL in the easy difficulty conditions (probably due to having less words to read and the same time as MNL stimuli with more words to read) disappears the harder the stimuli get with the exception of mid-way difficulty level 4.

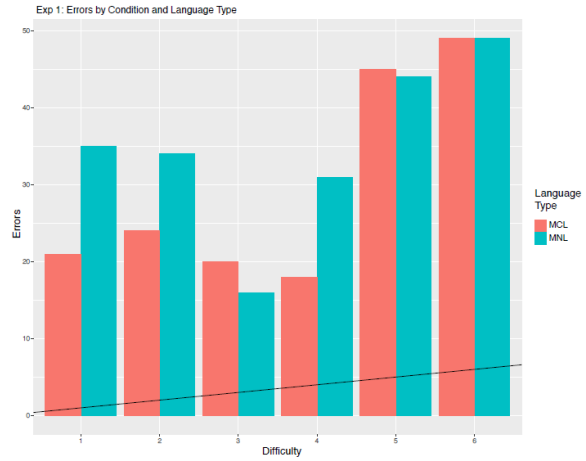


Figure 6, Histogram of errors in MNL and MCL in the 6 difficulty conditions

3. Did the language factor play a different role for the different types of congruency responses regarding reaction times? It was important to verify whether there was an effect of congruent stimuli versus incongruent stimuli (to the corresponding image) since congruent stimuli were deemed easier targets than incongruent ones, therefore understanding incongruent stimuli constitutes an extra difficulty condition in and of itself. To illustrate this with a concrete example: An image that shows an empty parking lot with a message that reads "Parking is available" is easier to interpret as a "yes congruent" than an image showing a desk lamp with a message that reads "Ceiling lamp is available" as a "no, incongruent". Confusion might arise from the presence of a lamp in the picture but which is not a ceiling lamp. Most incongruent images were purposefully chosen to include a little forced ambiguity, or an extra "trick" where the participant had to verify thoroughly the image before responding. Therefore, we compared the general effect of reaction times regardless of difficulty for congruent stimuli in both language conditions using the Wilcox signed Rank test. There was no significant difference in reaction times of the congruent stimuli produced in the MCL (Median = 1888.502 ms) and MNL (Median = 1879.167 ms) conditions; v = 1468, p = 0.3875. However, when performing the same test for the incongruent stimuli we found a significant difference in the MCL (Median = 2241.473ms) and the MNL (Median = 1927.541ms) conditions; v = 1475, p = 0.0308. As we can see from Table 2 the difference between medians in the incongruent condition is far superior than the congruent one and is statistically significant. We attribute this difference to the added difficulty in the interpretation of the incongruent stimuli, and we conclude that the MNL syntax helps process information faster than the MCL condition as the difficulty in the task and stimuli increase.

MCL Congruent	MNL Congruent	Difference	MCL Incongruent	MNL Incongruent	Difference
1888.502	1879.167	9.335	2241.473	1927.541	+313.932

Figure 7, Medians in ms of MCL and MNL reaction times in congruent and incongruent stimuli

4. Did the language factor play a different role for different levels of English placement (Basic Intermediate, Mastery, Natives) regarding reaction times?

We grouped the English placement levels into 3 categories. "Basic intermediate" regroups participants that were placed from levels A2 to C1, "Mastery" has participants that were placed in C2 level and "native" are the native English speaker participants. We did a series of t-tests (as reaction times for those sub-groups were not significantly non-normal so we could use a parametric test) to compare the two different language conditions in each of the English placement groups. For basic intermediate level, there was a significant difference in the scores for MCL (Mean = 2246.322 ms) and MNL (Mean = 2144.104 ms) conditions; t = 2.5416, p = 0.01644. For mastery level, there was no significant difference in the scores for MCL (Mean=1956.563ms) and MNL (Mean= 1954.745ms) conditions; t = 0.034395, p = 0.9728. For native level, there was no significant difference in the scores for MCL (Mean = 1690.904 ms) and MNL (Mean = 1588.062 ms) conditions; t = 1.8301, p = 0.09444. As we can see the only significant result is the basic intermediate level. We can conclude that MNL helps comprehension for the weaker levels of English levels as reaction times are significantly shorter for that group. While the native group does not show statistical significance, most probably because the group is made up of 12 participants only, it is interesting to note the difference in the average of the MNL and MCL which is equal to the difference for lower intermediates (averages which showed statistical significance). Native speakers often mentioned that they preferred the more natural language, and this is also apparent

in their results. A simple linear regression was also calculated to predict the reaction times of the MCL responses based on the 3 English placement levels. A significant regression equation was found (F(2,432) = 21.83, p = 9.275e-10), with an R2 of 0.0918. Participants' predicted reaction times is equal to 2221.92 – 280.14 ms for every English placement level gained. Therefore, reaction time decreased 280.14 ms for every English placement level gained (cf. Figure 8).

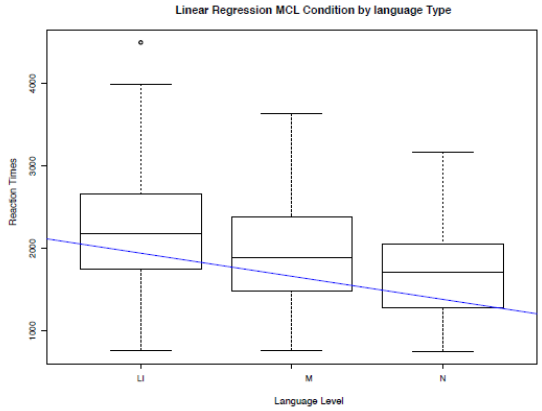


Figure 8, Linear regression of MCL in the different English Placement Levels

A simple linear regression was calculated to predict the reaction times of the MNL responses based on the 3 English placement levels. A significant regression equation was found (F(2,430) = 21.38, p = 2.288e-10), with an R2 of 0.0981. Participants' predicted reaction times is equal to (2146.50 ms – 190.20 ms) for every English placement level gained. Therefore, reaction times decreased 190.20 ms for every English placement level gained (cf. Figure 39).

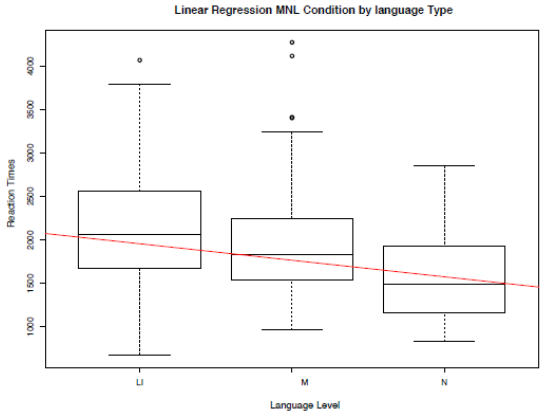


Figure 9, Linear regression of MNL reaction times in the different English Placement Levels

A graphical representation of both of those linear regressions is shown in Figure 10. As we can see, there is no interaction between these two languages for all three English level placements, but they both show decreasing reaction times with every additional level of English placement. The MNL proves to have consistently faster reaction times in all English placement levels, and therefore, we can conclude that MNL helps comprehension and information processing more than MCL regardless of participants' English placement level.

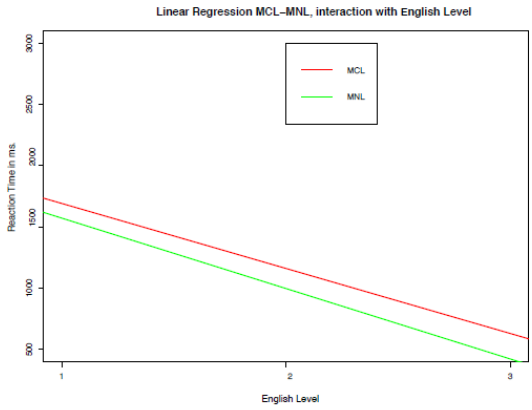


Figure 10, Linear regression for both MCL and MNL reaction times in the different English Placement Levels

4 Discussion

As shown in the results of hypothesis 1, MNL condition shows significantly faster reaction times than MCL condition, and both languages performed equally with regards to accuracy (in hypothesis 2). This could be explained by the fact that the syntactic changes (sentential elements in constative statements) between the two language conditions did not have enough disparities to warrant observable differences in accuracy, whereas the observed differences in reaction times were able to highlight the subtle syntactic variations that led to faster comprehension. In the experiment speed of stimuli presentation and to a certain degree the stress it provoked, accentuated the role of the more natural language in information processing. Additionally, there was no interaction between the two languages with regards to the 6 levels of syntactic difficulty, but reaction times get slower when difficulty increases in both languages. We can also conclude that MNL produced consistently faster reaction times than MCL in all syntactic difficulty

conditions. As we illustrated in research question 3, incongruent stimuli had an additional touch of difficulty and that is reflected in the reaction times' discrepancies for congruency conditions in both language conditions. Incongruent stimuli showed significantly faster reaction times for the MNL condition over the incongruent MCL condition, while the congruent stimuli did not. Therefore, in cases of increased difficulty the more natural language helps ease comprehension. Concerning English placement levels (research question 4), MNL seems to facilitate comprehension for participants in the basic intermediate level placement, and this suggests that speakers with weaker levels of English proficiency would benefit more greatly from a more natural language than confirmed speakers, or at least we could say that the effect is more conspicuous. While native English speakers performed better on average in the MNL condition, the effect was not statistically significant and should be the object of further studies with bigger samples of native speakers. We could also conclude that there is no interaction between the reaction time of the two language conditions and the different English level placement (one language did not start out having better performance than the other but ended up performing worse in different level placements), however we do observe a downward tendency in reaction times the more proficient speakers become. Natives have significantly faster reaction times than basic intermediate English speakers.

5 Conclusion

We presented in this experiment a congruency task similar to traditional judgment tasks in behavioral experiments. It provided a firmly controlled environment to test linguistic hypotheses and CNL rules, nonetheless, the downside of using such experiments is that we are limited to evaluating passive comprehension, mainly of specific informative statements. It would be quite difficult to evaluate the comprehension of an order or an instruction using traditional judgment tasks. In subsequent experiments, the congruency tasks will be replaced by ecological performance tasks for injunctive statements (participants performed the action required and the accuracy and response times are recorded) which include the urgency factor (speed of stimuli, and stress generated by limited response time). We will also be recruiting more native speaker participants to have a more representative panel of the target population (pilots from all around the globe), and ascertain whether the different syntactic language conditions reflect equally on native and non-native English speakers.

The results from this experiment are somewhat satisfactory as they show that our initial hypothesis is validated in a certain number of conditions. In all cases, contrary to common misconceptions, results showed that more simplification and linguistic economies and ellipses hardly ever led to better performance (MCL conditions did in no condition show significantly better reaction times or accuracy than MNL conditions). Furthermore, this experiment brought us first elements of empirically tested data which question controlled language construction, and the limits of simplification in general. It showed that what we sometimes mistakenly label as superfluous or empty syntactical elements (such as grammatical words as opposed to lexical words) could go a long way in ensuring better comprehension and faster information processing from a psycho-linguistic point of view.

References

Aikawa, T., Schwartz, L., King, R., Corston-Oliver, M., Lozano, C. 2007. *Impact of Controlled Language on Translation Quality and Post- Editing in a Statistical Machine Translation Environment*. Proceedings of the MT Summit XI, 1-7.

Cardon, R., Grabar, N. 2018. *Identification of Parallel Sentences in Comparable Monolingual Corpora from Different Registers.*. In Proceedings of the Ninth International Workshop on Health Text Mining and Information Analysis (pp. 83-93).

Condamines, A., Warnier, M. . 2014. *Linguistic Analysis of Requirements of a Space Project and Their Con-formity with the Recommendations Proposed by a Con-trolled Natural Language.*. In International Workshop on Controlled Natural Language (pp. 33-43). Springer International Publishing.

Condamines, A. 2021. *Towards an ergonomic linguistics: Application to the design of controlled natural languages.*. International Journal of Applied Linguistics, 31(1), 18-30.

Flesch, R. 1979. *How to Write Plain English: Let's Start with the Formula*. University of Canterbury.

ISO. 2016. *Ergonomics of human-system interaction: part 11: usability: definitions and concepts*

(ISO/DIS 9241-11.2:2016). German and English version prEN ISO 9241-11:2016.

Jahchan, N., Condamines, A., Cannesson, E. 2016. *To What Extent Does Text Simplification Entail a More Optimized Comprehension in Human-Oriented CNLs?*. In International Workshop on Controlled Natural Language (pp. 69-80). Springer International Publishing.

Jahchan, N. 2017. *The importance of using psycholinguistic tools for CNL evaluations*. Actes de Jetou, 99-105.

Jahchan, N. 2019. *To what extent does text simplification improve human comprehension?: cognitive evaluations for the optimization of the Airbus cockpit controlled language for future aircraft*. (Doctoral dissertation, Universite Toulouse le Mirail-Toulouse II).

Kittredge, R. I.. 2003. *Sublanguages and controlled languages*. In Ruslan Mitkov, editor, The Oxford Handbook of Computational Linguistics (pp. 430-447).

Kuhn, T. 2014. *A Survey and Classification of Controlled Natural Languages. Computational Linguistics*. 40(1) (pp. 121-170).

Kuhn, T. 2010. *An Evaluation Framework for Controlled Natural Languages*. In Norbert E. Fuchs, editor, Proceedings of the Workshop on Controlled Natural Language (CNL 2009), Lecture Notes in Computer Science 5972 (pp. 1-20). Springer, 2010.

Leroy, G., Helmreich, S., Cowie, J. R. 2010. *The effects of linguistic features and evaluation perspective on perceived difficulty of medical text*. In 43rd Hawaii International Conference on System Sciences (pp. 1-10). IEEE.

O Brien, S.. 2003. *Controlling Controlled English: An Analysis of Several Controlled Language Rule Sets*. In Proceedings of EAMT-CLAW, 3 (pp. 105-114), 33.

O Brien, S., Roturier, J. . 2007. *How Portable Are Controlled Language Rules? A comparison of Two Empirical MT Studies.* In Proceedings of MT summit XI (pp. 345-352).

Schwitter, R. 2010. *Controlled Natural Languages for Knowledge Representation*. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters (pp. 1113-1121). Association for Computational Linguistics.

Spaggiari, L., Beaujard, F., Cannesson, E. 2003. A Controlled Language at Airbus. *Proceedings of EAMT-CLAW03 (pp. 151-159)*.

Szmrecsanyi, B. 2004. *On operationalizing Syntactic Complexity. In Le poids des mots.*. Proceedings of the 7th International Conference on Textual Data Statistical Analysis. Louvain-la-Neuve Vol. 2 (pp. 1032-1039)

Tanguy, L., Tulechki, N.. 2009. *Sentence Complexity in French: A Corpus-based Approach.* In Intelligent information systems (IIS) (pp. 131-145).

Trauzettel-Klosinski, S., Dietz, K. 2012. *Standardized Assessment of Reading Performance: the New International Reading Speed Texts IReST.* . Investigative ophthalmology visual science, 53(9) (pp. 5452-5461)

Warnier, M. . 2018. *Contribution de la linguistique de corpus a la constitution de langues controlee pour la redaction technique : l exemple des exigences de projets spatiaux.*. Ph.D. thesis, University of Toulouse - Jean Jaures.

Grammar-Based Concept Alignment for Domain-Specific Machine Translation

Arianna Masciolini
Digital Grammars
arianna@digitalgrammars.com

Aarne Ranta
University of Gothenburg,
Department of Computer Science
and Engineering;
Digital Grammars
aarne.ranta@cse.gu.se

Abstract

Grammar-based domain-specific MT systems are a common use case for CNLs. High-quality translation lexica are a crucial part of such systems, but involve time consuming work and significant linguistic knowledge. With parallel example sentences available, statistical alignment tools can help automate part of the process, but they are not suitable for small datasets and do not always perform well with complex multiword expressions. In addition, the correspondences between word forms obtained in this way cannot be used directly. Addressing these problems, we propose a grammar-based approach to this task and put it to test in a simple translation pipeline.

1 Introduction

Grammar-based translation pipelines such as those based on Grammatical Framework (GF) have been successfully employed in domain-specific Machine Translation (MT) (Ranta et al., 2020). What makes these systems well suited to the task is the fact that, when we constrain ourselves to a specific domain, where precision is often more important than coverage, they can provide strong guarantees of grammatical correctness.

However, lexical exactness is, in this context, just as important as grammaticality. An important part of the design of a Controlled Natural Language (CNL) is the creation of a high-quality translation lexicon, preserving both semantics and grammatical correctness. A translation lexicon is often built manually, which is a time consuming task requiring significant linguistic knowledge. When the task is based on a corpus of parallel example sentences, part of this process can be automated by means of statistical word and phrase alignment techniques (Brown et al., 1993; Och and Ney, 2000; Dyer et al., 2013). None of them is, however, suitable for the common case in which only a small amount of example data is available — typically, with just one occurrence of each relevant lexical item.

In this paper, we propose an alternative approach to the automation of this task. While still being data-driven, our method is also grammar-based and, as such, capable of extracting meaningful correspondences even from individual sentence pairs.

A further advantage of performing syntactic analysis is that we do not have to choose *a priori* whether to focus on the word or phrase level. Instead, we can simultaneously operate at different levels of abstraction, extracting both single- and multiword, even non-contiguous, correspondences. For this reason, we refer to the task our system attempts to automate as *Concept Alignment* (CA). A *concept* is a semantic unit expressed by a word or a construction, which is also a unit of *compositional translation*, where translation is performed by mapping concepts to concepts in a shared syntactic structure.

Conceiving concepts as *lemmas* equipped with morphological variations rather than fixed word forms or phrases allows us to generate translation lexica complete with grammatical category and inflection, so that correct target language forms can be selected in each syntactic context.

This paper is structured as follows. Section 2 starts by giving an overview of our approach to CA and comparing it with related work, followed by a description of our CA algorithm. Section 3 presents the results obtained in a first evaluation of the system. Section 4 summarizes our conclusions and discusses some ideas for future work.

2 Methodology

The objective of CA is to find semantical correspondences between parts of multilingual parallel texts. We call *concepts* the abstract units of translation, composed of any number of words, identified through this process, and represent them as *alignments*, i.e. tuples of equivalent concrete expressions in different languages.

The basic use case for CA, which we refer to

specifically as *Concept Extraction* (CE), is the generation of a translation lexicon from a multilingual parallel text. This is analogous to the well-known earlier word and phrase alignment techniques.

An interesting and less studied variant of CA is *Concept Propagation* (CP), useful for cases where a set of concepts is already known and the goal is to identify the expressions corresponding to each of them in a new language, potentially even working with a different text in the same domain. While our system does implement basic CP functionalities, in this paper we focus on its most mature portion: CE. Because results analogous to those that could be obtained via multilingual extraction can be obtained more easily with a combination of CE and CP, we restrict ourselves, for the time being, to bilingual corpora.

As stated in the Introduction, most existing alignment solutions are based on statistical approaches and are, as a consequence, unsuitable for small datasets. Grammar-based approaches, making use of parallel treebanks and collectively referred to as *tree-to-tree alignment methods*, have also been proposed (Tiedemann, 2011), but have historically suffered from the inconsistencies between the formalisms used to define the grammars of different languages and from the lack of robustness of parsers. This work is a new attempt in the same direction, enabled by two multilinguality-oriented grammar formalisms developed over the course of the last 25 years: Grammatical Framework (GF) (Ranta, 2011) and Universal Dependencies (UD) (Rademaker and Tyers, 2019).

GF is a constituency grammar formalism and programming language in which grammars are represented as pairs of an *abstract syntax*, playing the role of an interlingua, and a set of *concrete syntaxes* capturing the specificities of the various natural languages. In the case of translation, similarly to what happens in programming language compilation, strings in the source language are *parsed* to Abstract Syntax Trees (ASTs), which are then *linearized* to target language strings.

UD, on the other hand, is a dependency grammar formalism meant for cross-linguistically consistent grammatical annotation. As opposed to constituency, *dependency* is a word-to-word correspondence: each word is put in relation with the one it depends on, called its *head*, via a directed labelled link specifying the syntactic relation between them. Importantly for our application, the standard format for UD trees, CoNNL-U, gives information not only on the syntactic role of each word, but also on its Part-Of-Speech (POS) tag, lemma, and morphological features.

While both formalisms independently solve the issues related to having to work with grammars that are inconsistent with each other, UD is especially appealing since, being dependency trees an easier target, several robust parsers, such as (Straka et al., 2016) and (Chen and Manning, 2014) are available. Alone, UD trees are sufficient to extract (or propagate) tree-to-tree alignments, but not to automate the generation of a morphologically-aware translation lexicon for a generative grammar. This is where GF comes into play: after correspondences are inferred from a parallel text, our system is able to convert them to GF grammar rules, easy to embed in a domain-specific grammar but also making it immediate to carry out small-scale translation experiments using pre-existing grammatical constructions implemented in GF's Resource Grammar Library (RGL), which covers the morphology and basic syntax of over 30 languages. This is enabled by `gf-ud`, a conversion tool described in (Kolachina and Ranta, 2016) and (Ranta and Kolachina, 2017). Concretely, then, the system we propose consists of a UD parser, an alignment module based on UD tree comparison and a program, based on `gf-ud`, that converts them into the rules of a GF translation lexicon.

2.1 Extracting concepts

The core part of the system outlined above is the alignment module. Its function is to extract alignments from parallel bilingual UD treebanks. The outline of the algorithm is given in the following pseudocode:

---

procedure EXTRACT($criteria$,(t, u))
  $alignments = \emptyset$
  if (t, u) matches any alignment $criteria$ **then**
    $alignments$ += (t, u)
    for (t', u') in SORT(SUBTS(t)) × SORT(SUBTS(u))
do
      extract($criteria$,(t', u'))
  return $alignments$

Here, the input consists of a list of priority-sorted *alignment criteria*, i.e. rules to determine whether two dependency trees should be aligned with each other, and a pair (t, u) of UD trees to align. An example alignment criterion is sameness of syntactic label, which makes it so that, for instance, subjects are aligned with subjects and objects with

objects; the details will be discussed in Section 2.1.1. From an implementation point of view, UD trees are rose trees (trees with arbitrary numbers of branches) where each node represents a word with its dependents as subtrees (see Figure 1). The rose tree is easily obtained from the CoNLL-U notation that UD parsers produce.

As a first step, the program checks whether the two full sentence trees can be aligned with each other, i.e. if they match one or more alignment criteria. In the case of the example criterion discussed above, this means that their roots are labelled the same. If this is the case, they are added to a collection of alignments, which are represented as pairs of UD (sub)trees associated with some metadata, such as the id of the sentence they were extracted from. Such a collection is what the function will return after aligning all the dependency subtrees. The same procedure is applied recursively to all pairs of immediate subtrees of each sentence, until the leaves are reached or alignment is no longer possible due to lack of matching criteria. Subtrees are sorted based on their dependency label to give higher priority to pairs whose heads have the same label (cf. SORT in the pseudocode).

A simple but useful refinement is that, depending on which alignment criteria a pair of trees matches, the heads of the two trees may or may not also be added to the collection of alignments. This is done in order not to miss one-word correspondences that cannot be captured in any other way, for instance between the root verbs of two full sentences. A relevant implementation detail is that, in this context, the head of a tree is not simply defined as its root. Instead, if the root is part of a compound written as two or more separate words or a verb with auxiliaries, the root nodes of the corresponding subtrees are also considered parts of it.

When working on multiple sentences, the algorithm can be applied in an iterative fashion, so that knowledge gathered when a sentence pair is aligned can be used when working on later sentences and to keep track of the number of occurrences of each alignment throughout the entire text. Furthermore, it is possible to initialize the algorithm with a nonempty set of alignments, obtained with the same program or by means of a statistical tool outputting alignments in Pharaoh format and to combine the results of several extraction processes into a single translation lexicon.

2.1.1 Alignment criteria

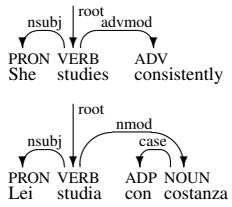
While the alignment criteria are customizable, to allow for a better understanding of the extraction algorithm described above, we explain the criteria that our implementation utilizes by default.

Matching UD labels The most obvious, but also most effective idea is to determine alignability based on comparing the dependency labels of the members of the candidate UD tree pair. In particular, according to this idea, two subtrees in *matching context*, i.e. attached to aligned heads, constitute an alignment if their roots share the same dependency label, meaning that they are in the same syntactic relation with their heads. Note that, since the root of a UD tree is always attached to a fake node with an arc labelled `root`, this criterion also implies that full sentences are always considered to align with each other. This is desirable since we assume that the parallel texts that are fed to our program are sentence-aligned.

Part-Of-Speech equivalence As noted above, the CoNNL-U notation provides information on the grammatical categories of each word, represented as Universal POS tags (Petrov et al., 2012). Intuitively, if the nodes of two trees in matching contexts have the same POS tags, the two trees are more likely to correspond to each other than if not. This is especially true if we focus, for instance, solely on the open class words (defined as in the UD documentation[1]), thus ignoring function words such as prepositions, determiners and auxiliary verbs, which tend to behave differently across different languages. A useful relation to define between dependency trees is, then, that of *POS-equivalence*: two dependency trees t_1, t_2 are POS-equivalent if $M_1 = M_2 \neq \emptyset$, where M_i is defined as the multiset of POS tags of all the open class word nodes of t_i. Applied as a backup for label matching, this criterion can be used to capture correspondences that would otherwise be missed, thus increasing recall, but a decrease in precision is also to be expected. However, since alignment criteria are defined as boolean functions, it is easy to combine them so to that they have to apply simultaneously. This can be useful in cases where precision is more important than recall.

Known translation divergence Parallel texts often present significant, systematic cross-linguistic

[1] universaldependencies.org/u/pos/all.html




```
1 She PRON 2 nsubj
2 studies VERB 0 root
3 consistently ADV 2 advmod

1 Lei PRON 2 nsubj
2 studia VERB 0 root
3 con ADP 4 case
4 costanza NOUN 2 obl
```

```
2 studies VERB 0 root
  3 consistently ADV 2 advmod
  1 She PRON 2 nsubj

2 studia VERB 0 root
  1 Lei PRON 2 nsubj
  4 costanza NOUN 2 obl
    3 con ADP 4 case
```

Figure 1: The graphical, simplified CoNNL-U and sorted rose tree representation of a pair of UD sentences. With the default criteria, which among other things allow for matching adverbial with adjectival modifiers, the resulting alignments are: ⟨*She studies consistently, Lei studia con costanza*⟩ (matching `root` label), ⟨*studies, studia*⟩ (head alignment), ⟨*she, lei*⟩ (matching `nsubj` label) and ⟨*consistently, con costanza*⟩ (translation divergence; `amod` and `advmod` treated as equivalent).

grammatical distinctions. When this is the case, it is often straightforward to define alignment criteria based on recognizing the corresponding patterns. While many distinctions of this kind are specific to particular language pairs or even stylistical, some of them occur independently of what languages are involved and do not depend on idiomatic usage nor aspectual, discourse, domain or word knowledge. Drawing inspiration from (Dorr, 1994), we refer to them as *translation divergences* and handle some of the most common ones explicitly. For instance, *categorial divergences* occur when the POS tag of a word in the source language changes in its translation. An ubiquitous example of this is that of adverbial modifiers (with the UD label `advmod`) translated as prepositional phrases (with the UD label `obl`, for *oblique*), such as in the English-Italian pair ⟨*She studies* ***consistently****, Lei studia* ***con costanza***⟩ (see Figure 1). *Structural divergences*, where a direct object in one language is rendered as an oblique in the other, as in the English-Swedish pair ⟨*I told* ***him****, Jag berättade* ***för honom***.⟩), are also frequently encountered.

Known alignment Another case in which it is trivially desirable for two subtrees in matching context to be aligned is when an equivalent alignment is already known, for instance due to a previous iteration of the extraction algorithm. When referred to pairs of alignments, the term *equivalent* indicates that the two alignments, linearized, correspond to the same string.

At a first glance, this might look like a criterion with no practical applications. However, it can be useful when, instead of starting with an empty set of correspondences, we initialize the program with some alignments that are either inserted manually or, most interestingly, obtained with some other alignment technique. For instance, in this way it is possible to combine the system proposed in this paper with a statistical tool and give more credit to correspondences identified by both.

2.1.2 Pattern matching

So far, we have described how CE can be used in a setting where the objective is to generate a comprehensive translation lexicon based on set of example sentence pairs. We pointed out that the program can be configured to prioritize precision or coverage, but we never restricted our search to a particular type of alignments. However, there are cases in which only certain syntactic structures are of interest: for instance, we might be looking for adverbs or noun phrases exclusively.

To handle such cases, the CE module can filter the results based on a `gf-ud` *tree pattern*. `gf-ud` supports in fact both simple pattern matching, which is integrated in the CE module itself, and pattern replacement[2]. Combining them, for instance by pruning the UD trees extracted by the alignment module, allows us to extract correspondences that cannot be identified by CE alone.

For example, pattern matching can extract verb phrases by looking for full clauses and dropping the subtrees corresponding to subjects. By means of replacements, one can obtain *predication patterns*, i.e. correspondences that specify the argument structure of verbs, such as the following English-Italian one:

⟨X *gives* Y Z, X *dà* Z *a* Y⟩.

2.2 Generating grammar rules

The alignments outputted by the CE module described so far are represented as pairs of UD (sub)trees in CoNLL-U format. While converting

[2]documentation is available at `github.com/GrammaticalFramework/gf-ud`

them to GF ASTs is one of `gf-ud`'s core functionalities, such trees also need be converted into the grammar rules of a compilable GF translation lexicon. To this end, our grammar generation module requires a *morphological dictionary* of the languages at hand and an *extraction grammar*.

Morphological dictionaries, implemented for several languages as part of the RGL, are large collections of lemmas associated with their inflectional forms.

An extraction grammar, on the other hand, defines the syntactic categories and functions the entries of the automatically generated lexicon are built with. For example, entries can be prepositional phrases (PP) or verb phrases (VP) constructed by the following GF functions:

```
PrepNP : Prep -> NP -> PP # case head
PrepPP : VP   -> PP -> VP # head obl
```

The dependency labels appended to the function types instruct `gf-ud` to build GF trees from UD trees that match these labels.

The final translation lexicon, i.e. a GF grammar that extends the extraction grammar, is then derived from these GF trees. In its abstract syntax, the name of each concept is associated with its grammatical category, i.e. the category of the root of the GF tree. For example:

```
fun in_the_field__inom_område_PP : PP ;
```

The concrete syntaxes, on the other hand, contain the linearization rules for each concept, directly based on the trees obtained from `gf-ud`. For instance, in English:

```
lin in_the_field__inom_område_PP =
  PrepNP
    in_Prep
    (DetCN the_Det (UseN field_N)) ;
```

Most function words, such as `in_Prep`, and many content words, such as `field_N`, are available through the morphological dictionaries. When this is not the case, they are assumed to be regular and an additional rule is generated for them. For instance, if the English morphological dictionary didn't contain the word "*field*", we would have:

```
oper field_N = mkN "field" ;
```

3 Evaluation

In this section, we evaluate the system proposed above. We first discuss the data used in the evaluation. After that, we describe our experiments, aimed at putting both the CE module *per se* and the entire system from parsing to lexicon generation to the test, and present our results.

3.1 Data

Because we want part of our evaluation to be independent from the quality of UD parsing, some of the experiments are carried out on treebanks instead of raw text. To this end, we use a 100-sentence subset of the Parallel UD (PUD) corpus, a set of 1000 manually annotated or validated sentences in CoNLL-U format. Of the over 20 languages PUD treebanks are available in, we selected English, Italian and Swedish. Using this data limits the amount of alignment errors that are due to annotation issues to a minimum, even though a small number of inconsistencies is present even in this corpus.

When it comes to testing the program on raw text, we use two small (< 1000 sentences) bilingual sentence-aligned corpora consisting of course plans from the Department of Mathematics and Computer Science (DMI) of the University of Perugia (for English-Italian) and from the Department of Computer Science and Engineering (CSE) shared between the University of Gothenburg and the Chalmers University of Technology (for English-Swedish). For brevity, we will refer to these two datasets as to the DMI and the CSE corpora. This data, available in the project repository, was collected and sentence-aligned specifically for this work and a related Bachelor's thesis project (Eriksson et al., 2020). When using raw text, our parser of choice is UDPipe (Straka et al., 2016). In particular, we use the ParTUT English and Italian models for the DMI corpus and models trained on the bilingual LinES English-Swedish treebank for the CSE corpus [3].

3.2 Evaluating CE

While we focus mostly on the MT applications of CA, automatic translation, and much less GF-based domain-specific translation, is not the only context in which CA can be put to use. For instance, it is easy to imagine using it to build translation memories to be use as an aid for human translation. For this reason, a first set of experiments is aimed at evaluating the alignments obtained with our CA module independently from the other stages of our lexicon generation pipeline.

[3] The pretrained UDPipe models and information about their performance are available at `ufal.mff.cuni.cz/udpipe/1/models`

	CE		fast_align (100 sentences)		fast_align (full dataset)	
	en-it	**en-sv**	**en-it**	**en-sv**	**en-it**	**en-sv**
distinct alignments	536	638	1242	1044	1286	1065
correct	392 (73%)	514 (80%)	346 (28%)	538 (52%)	540 (42%)	677 (64%)
usable in MT	363 (68%)	503 (79%)	316 (25%)	525 (50%)	510 (40%)	666 (63%)

Table 1: Comparison between our grammar-based CE module and `fast_align` on PUD data, training the statistical model both on 100 and 1000 sentences and discarding the alignments obtained for sentences 101-1000 in the latter case.

We first assess the correctness of the alignments the CE module is able to extract from the PUD treebanks, comparing our results with those obtained with a statistical tool, `fast_align` (Dyer et al., 2013). In addition, we try to quantify the impact of automated UD parsing on the performance of the CE module by comparing the above results with those obtained on the DMI and CSE corpora.

While precision and recall are two well-known performance metrics, the lack of a gold standard for CE forces us to, before calculating the ratio between the number of correct alignments and the total number of extracted alignments, manually assess the correctness of each alignment. What is more, since some alignments are only correct in the specific context of the sentence pair in which they occur, we make a further distinction between correct alignments that are relevant for a translation lexicon and alignments that are useful for comparing the sentences but should not be used for MT. As an extreme example of a pair-specific alignment, consider the sentences ⟨*He missed the boat, Ha perso il treno*⟩. In both languages, the idea of missing a chance is expressed with idiomatic expressions very similar to each other. However, the Italian translation mentions a train ("*treno*") in the place of a boat.

3.2.1 Results on manually annotated treebanks

In Table 1, we compare the results obtained with our grammar-based module to those obtained statistically on the PUD treebanks. Of course, `fast_align` does not make use of the information present in the CoNLL-U files except with regards to tokenization. On the other hand, the relatively large size of the PUD treebanks makes it possible also to train the statistical tool on the full dataset instead of just using the chosen 100-sentences subset, allowing for a fairer comparison. In both cases, `fast_align` is run with the recommended parameters and the CE program is configured to only extract one-to-many and many-to-one word alignments, as `fast_align` does not align larger phrases. This explains CE's seemingly low recall. To get a better idea, Table 1 can be compared with Table 2, which summarizes the results of an experiment where no constraints are placed on the size of the extracted alignments.

While `fast_align` is designed to align every word in the text (or explicitly state that a word has no counterpart in its translation) and, consequently, extracts around twice as many correspondences, the percentage of correct correspondences is definitely in favor of our system, even though `fast_align` gets significantly more precise when trained on the full 1000-sentence dataset.

3.2.2 Results on raw text

The course plan corpora are significantly harder to work with, the additional challenges being the inexactness of many translations (which is the direct cause of some of the alignment errors encountered in our evaluation) and the fact that our CE module relies, in this case, on the quality of automatic lemmatization, POS-tagging and parsing.

In Table 2, we compare the results obtained on manually annotated data and the course plans corpora parsed with UDPipe.

What is immediately evident, but not unexpected, is a decrease in precision. The percentage of correct alignments, however, stays significantly higher than that obtained with `fast_align` in the previous experiment, even with the model trained on the full PUD corpus. In fact, even though percentages seem similar for English-Swedish, the CSE corpus is roughly half the size of the full PUD corpus.

The results are less encouraging in terms of recall: the number of alignments extracted from the course plan corpora is similar to that obtained from the PUD treebank sample, despite the difference in size. This is explained in part by the presence, in the course plans corpora, of a large amount of very short sentences, and in part by the fact that parse

	PUD (100 sentences)		course plans	
	en-it	en-sv	DMI (881 sentences)	CSE (539 sentences)
distinct alignments	1197	1325	1823	1950
correct	916 (77%)	1112 (85%)	1205 (66%)	1296 (66%)
usable in MT	880 (74%)	1099 (84%)	1157 (63%)	1248 (64%)

Table 2: Comparison between the grammar-based extraction of alignments of any size from manually annotated PUD treebanks and from automatically parsed sentences from the course plans corpora.

errors introduced by UDPipe make it impossible to align many subsentences without a significant loss in terms of precision.

Our system is, on the other hand, capable of extracting multiword alignments that are unlikely to be identified by a statistical tool, especially in the case of such a small dataset. Examples of this are the noun phrases ⟨*the aim of the course, l'obiettivo del corso, syftet med kursen*⟩ (a concept found in both corpora and, as such, trilingual), ⟨*Natural Language Processing, språkteknologi*⟩ and ⟨*object oriented programming, programmazione ad oggetti*⟩.

3.3 MT experiments

The second set of experiments has the objective of assessing the quality of the final output of the system we propose: GF translation lexica. Because we are now focusing on using CA in the context of domain-specific MT, we do not make use of the PUD treebanks, where sentences come from a variety of different sources, but just of the course plans corpora. We do not construct a grammar specific to such domain: for small-scale MT experiments, it is sufficient to extend the extraction grammar itself with preexisting syntax rules defined in the RGL.

The idea is to automatically translate a set of English sentences to Italian and Swedish, ask native speakers of the target languages to produce a set of reference translations, and compare them to the original machine-generated ones by computing BLEU scores. Due to the small size of the datasets and the consequently low coverage of the extracted lexicon, we generate the sentences to translate directly in the GF shell rather than trying to parse arbitrary sentences from other course plans. In order to do that, we make use of GF's random AST generation functionality but at the same time manually select semantically plausible sentences to facilitate the task of the human translators. The results of this process are two small testing corpora, one for the DMI and one for the CSE corpus, each consisting of 50 English sentences. Reference translations are obtained by asking participants to compare the original English sentences to their automatically translated counterparts and correct the latter with the minimal changes necessary to obtain a set of grammatically and semantically correct translations. This is important as, if the reference translations are obtained independently, BLEU scores can easily become misleading.

3.3.1 Results

Corpus-level BLEU scores for the automatic translations of the 50+50 sentences of the testing corpora are summarized in Table 3. Following conventions, we report the cumulative n-gram scores for values of n from 1 to 4 (BLEU-1 to BLEU-4). However, being a significant portion of the sentences of length 4 or less, we also report BLEU-1 to BLEU-3 scores, BLEU-1 to BLEU-2 scores and scores obtained considering unigrams only.

	DMI (en-it)	CSE (en-sv)
BLEU-1 to 4	55	61
BLEU-1 to 3	63	68
BLEU-1 to 2	70	74
BLEU-1	79	81

Table 3: BLEU scores for automatic translations based on the course plans grammars.

These synthetic figures are useful to give an idea of the general quality of the translations: overall, although with relatively low scores, English-to-Swedish translation works significantly better than English-to-Italian. Looking back at the results reported in Section 3.2.2, the reason for this is not immediately clear, as the difference in precision between the two language pairs is negligible in the course plan corpora.

Looking at sentence-level scores can, however, be more insightful. For both corpora, scores assigned to individual segments range from the minimum possible value of 0 to the perfect score of 100, which indicates a perfect correspondence between the automatic and reference translation. Examples of sentences that were assigned a perfect BLEU-1 to 4 score are "*the library provides useful textbooks*" (translated to Italian as "*la biblioteca fornisce libri utili*") in the DMI corpus and "*this lab is more dif-*

ficult than the exam" (whose Swedish translation is "*den här laborationen är svårare än tentamen*") in the CSE corpus. On the other hand, it is easy for shorter sentences to be assigned the minimum BLEU-1 to 4 score even when they only contain a single grammatical or semantic error.

Furthermore, a problem with using the BLEU score as the only evaluation metric is the fact that it makes no distinction between content and function words, thus not allowing an evaluation focused specifically on the extracted concepts. The small size of the corpus, however, allows for some error analysis. From the participants' observations about the kind of errors encountered when manually editing the automatic translations, summarized in Table 4, we can conclude that while most errors are in fact due to wrong alignments, the main difference between two corpora lies in the number of translations that only contain grammatical errors. This explains the significant difference observed in the cumulative BLEU scores shown in Table 3.

	DMI (en-it)	**CSE (en-sv)**
semantical	23 (46%)	23 (46%)
grammatical	10 (20%)	3 (6%)
both	3 (6%)	4 (8%)

Table 4: Types of errors encountered in the automatically translated sentences.

Among other things, many Italian contractions such as "*del*" ("*di*" + "*il*", in English "*of the*") are systematically rendered as two separate words due to UDPipe tokenization. Grammatical errors in Swedish are less common and less systematic. Only in one case, for instance, gender is incorrect ("*programbibliotek**en***"). These errors are easy to handle when writing a domain-specific grammar or, in cases like the latter, by making small adjustments to the morphological dictionaries.

Some errors regarding the extracted concepts are also interesting to analyze: the alignment ⟨*class, classe*⟩, for instance, causes the sentence *"I will attend the class"* to be (incorrectly) translated as *"io seguirò la classe"* instead of *"io seguirò la lezione"* even though the correspondence is in fact valid in most contexts in which (within the same domain!) *"class"* is not to be intended as a synonym of *"lesson"* but as teaching group.

4 Conclusions

We have presented a syntax-based alignment method with a focus on its applications in domain-specific translation lexicon generation. Compared with the existing statistical tools, our system has the following advantages:

- it performs consistently well even on small parallel corpora
- it is able to simultaneously extract correspondences between individual words, multiword expressions and longer phrases, including discontinuous constituents
- in conjunction with `gf-ud` pattern matching, it can be used to extract specific types of correspondences, such as predication patterns
- it can automatically generate compilable, morphology-aware GF translation lexica
- it can be configured to easily handle systematic, possibly language pair- or corpus-specific translation divergences.

While it requires manual corrections and completions to an extent that varies according to the quality of the data and the strictness of the chosen criteria, using the alignments obtained with our method can reduce the time required for bootstrapping the translation lexicon building process for a domain-specific CNL significantly. In fact, especially if a comprehensive morphological dictionary is available, part of the alignments will be ready to use in a GF-based system without any intervention.

The tangible fruits of this work are a Haskell library and a number of executables offering a user-friendly interface to perform CE, lexicon generation and various kinds of evaluations. The source code, including a preliminary implementation of CP, is available on GitHub[4]. The software has already been used to analyse customer-provided data in two commercial projects at Digital Grammars.

4.1 Current and future work

Our results, while encouraging, suggest that there is room for improvement in many different directions.

An obvious possible development is optimizing the current, initial implementation of Concept Propagation (CP) for its two use cases: propagating alignments to a new language looking for correspondences using a translation of the same text they were extracted from or using a different text in the same domain. An alternative to the former is to make CE, now working on bilingual texts, n-lingual.

[4] `github.com/harisont/concept-alignment`

When large enough amounts of data are available, using our system in conjunction with a statistical tool seems promising. As discussed above, this is already partially supported and it could prove useful to develop CA as an actual hybrid system.

Finally, since the freedom that generally characterizes human translation and the quality of currently available UD parsers make maximizing both alignment precision and recall unrealistic, tools to make it easier to postprocess the automatically generated lexica are under development.

References

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Bonnie J. Dorr. 1994. Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, 20(4):597–633.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

Eriksson, Gabrielsson, Hedgreen, Klingberg, Vestlund, and Ödin. 2020. Grammar-based translation of computer science and engineering terminology.

Prasanth Kolachina and Aarnte Ranta. 2016. From abstract syntax to Universal Dependencies. In *Linguistic Issues in Language Technology, Volume 13, 2016*. CSLI Publications.

Franz Josef Och and Hermann Ney. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 440–447, Hong Kong. Association for Computational Linguistics.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2089–2096, Istanbul, Turkey. European Language Resources Association (ELRA).

Alexandre Rademaker and Francis Tyers, editors. 2019. *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019)*. Association for Computational Linguistics, Paris, France.

Aarne Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford.

Aarne Ranta, Krasimir Angelov, Normunds Gruzitis, and Prasanth Kolachina. 2020. Abstract syntax as interlingua: Scaling up the grammatical framework from controlled languages to robust pipelines. *Computational Linguistics*, 46(2):425–486.

Aarne Ranta and Prasanth Kolachina. 2017. From Universal Dependencies to abstract syntax. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 107–116, Gothenburg, Sweden. Association for Computational Linguistics.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: Trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4290–4297, Portorož, Slovenia. European Language Resources Association (ELRA).

Jörg Tiedemann. 2011. Bitext alignment. *Synthesis Lectures on Human Language Technologies*, 4(2):1–165.

Tailoring a Controlled Language Out of a Corpus of Maintenance Reports

Yannis Haralambous
UMR 6285 Lab-STICC, DECIDE
& Département Informatique,
IMT Atlantique,
CS 83818,
29238 Brest Cedex 3
yannis.haralambous@imt-atlantique.fr

Tian Tian
UMR 6285 Lab-STICC, DECIDE
& Département Informatique,
IMT Atlantique,
CS 83818,
29238 Brest Cedex 3
tian.tian@imt-atlantique.fr

Abstract

We introduce a method for tailoring a controlled language out of a specialized language corpus, as well as for training the user to ensure a smooth transition between the specialized and the controlled language. Our method is based on the selection of maximal coverage syntax rules. The number of rules chosen is a naturalness vs. formality parameter of the controlled language. We introduce a training tool that displays segmentation into left-to-right maximal parsed sentences and allows utterance modification by the user until a complete parse is achieved. We have applied our method to a French corpus of maintenance reports of boilers in a thermal power station and provide coverage and segmentation results.

1 Introduction

The distinction between naturalist and formalist approach to controlled natural language has been widely discussed in the literature (Pool, 2006; Clark et al., 2010; Gruzitis et al., 2012; Marrafa et al., 2012; Kuhn, 2014). We will adopt an intermediate approach. Indeed, in this paper we deal with the specific problem of optimizing information and knowledge extraction out of utterances in a specialized but spontaneously written language, the language of maintenance reports of the boilers of a thermal power station. The reports are written under conditions of stress and lack of time, and therefore largely adopt a "telegraphic" spontaneous style, without post-editing or spell checking. The ultimate goal of our ongoing project is to mine the text in order to have it correlated with time-stamped data from sensors in the equipment, in search of anomalies. Use of a controlled language is expected to improve the text mining process, but asking technicians to rigorously adhere to a specific controlled language fragment is not an option.

We have therefore chosen to make a compromise by designing a controlled language based on the existing maintenance reports corpus vocabulary but with a restricted grammar. We have developed an editor that displays, in a non-intrusive way, success or failure of the syntax parse of written utterances. This means that we are adopting a naturalist approach while using tools from the formalist approach (subsets of syntax rules) to simplify the legacy natural language and make it easier to interpret.

To bring the controlled language closer to standard French, we used an additional, totally independent corpus, consisting of sentences in regular and carefully edited French. We extracted Phrase-Structure Grammar rules from this corpus.

Let us call $\mathcal{S}$ the maintenance report corpus and $\mathfrak{S}$ its set of production rules, $\mathcal{M}$ the regular French corpus and $\mathfrak{M}$ its production rules. By using the vocabulary of $\mathcal{S}$ and allowing only the most frequent elements of $\mathfrak{M}$ (and potentially some frequent rules from $\mathfrak{S}$) we define a controlled language that is a simplification of $\mathcal{S}$ (Saggion, 2017). Our first innovation is the possibility of tailoring the formality/naturalness of the controlled language by reducing/increasing the number of allowed production rules. Furthermore, the fact that these rules belong to a "golden corpus of standard French," entails a regularization of the informal (and syntactically chaotic) language of $\mathcal{S}$.

Syntax is very central to our approach because the vocabulary and its morphological variation are limited, due to the technical nature of the corpus.

Our second innovation is an editor with a "non-intrusive" training interface. A parsed utterance is displayed in blue color (or bold style, or some other graphical attribute) and the potentially unparsed part of it remains in standard style. Depending on working conditions during the text authoring act, the technician can choose to invest time and energy in "improving" eir[1] linguistic production, or ignore

[1] We use gender-neutral Spivak pronouns https://en.wikipedia.org/wiki/Spivak_pronoun.

the fact that the utterance has not been entirely parsed. When modifying the utterance, immediate feedback (by some graphical artefact) is provided to the author, who is thereby entering a smooth training process.

2 Related work

(Clark et al., 2010) define CPL and CPL-Lite as two variants of the same *Computer-Processable Language*, where

> While CPL searches for the best interpretation, CPL-Lite is simpler and interpreted deterministically (no search or use of heuristics). (Clark et al., 2010, 69)

In CPL-Lite, 113 sentence templates are allowed, giving rise to an equal number of binary predicates in Prolog-like syntax. We generalize this principle by allowing a variable number of production rules.

Despite the differences between the two languages, (Kuhn, 2014) considers only one CPL language and assigns it a PENS classification of $P^3E^3N^4S^2$FWI.[2] Other controlled languages based on a limited number of production rules are SQUALL ($\sim$ 50 rules) (Ferré, 2012), ucsCNL ($\sim$ 140 rules) (Barros et al., 2011) and Attempto ($\sim$ 360 rules without disjunctions) (Fuchs, 2018). These have been classified as $P^5E^2N^3S^4$FWA, $P^5E^2N^4S^4$FWDA and $P^4E^3N^4S^3$FWA, respectively by (Kuhn, 2014).

As we see, the lesser the rules (e.g., in SQUALL) the higher is P (precision). In our case the controlled language can be built with a variable number of production rules, so P can be variable, probably between P^2 and P^4. Expressiveness is rather low for the languages mentioned, but in our case this is of no concern since the application domain is very narrow: quantification is very sparse since maintenance reports concentrate on a small number of boilers and their parts, general rule structures are also limited since sentences are almost always declarative—only the presence of negation is mandatory, to express failure of equipment. As for naturalness, by using the same vocabulary as $\mathcal{S}$ and building syntax based on the rules of standard well-formed French, a high degree of naturalness is achieved, which we estimate around N^4. Simplicity can be assessed with more difficulty since there is no explicit description of the language. This description would imply giving and explaining all production and semantic rules involved and such a description can indeed be done but will not be provided to the language's users. Users are intended to adapt progressively to the controlled language—potentially a short notice on the editor's working principle may be addressed to them, but it will by no means be a comprehensive description of the language. We would therefore rather consider this language as S^2, a "language without exhaustive description," even though such a description would theoretically be possible. As for properties, these would be W (written) D (specific narrow domain) and I (industry).

[2] (Kuhn, 2014) defines letter codes for properties of controlled natural languages of different categories. In this frame, C stands for comprehensibility as goal of the language; T for translation; F for formal representation. As for the form of the language, W stands for written languages and S for spoken ones; and D stands for languages in a specific narrow domain. As for origin of the controlled language, three codes are defined: A stands for languages originating from academia, I from industry and G from government. (Kuhn, 2014) defines the PENS classification scheme to describe controlled languages according to four axes: P (precision), E (expressiveness), N (naturalness) and S (simplicity). For each dimension, five degrees (arbitrarily) are used, such as $P^1E^5N^5S^1$ for standard English and $P^5E^1N^1S^5$ for propositional logic.

3 Description of the Corpora

Our main corpus $\mathcal{S}$ consists of 2,280 maintenance reports, written in 8-hour intervals during two years. The volumetry of $\mathcal{S}$ is as follows: 30,851 sentences, 138,140 words. We explore its properties in Sections 4 (lexicon), 5 (morphology) and 6 (syntax).

To serve as a "ground truth" of French syntax, we built a second corpus, $\mathcal{M}$, based on eleven Harlequin-like novels by a well-known author. They are written in informal everyday French language, carefully edited by the publisher since the given novels are best-sellers with a very large audience. We have parsed the two corpora using the Stanford CoreNLP parser and have kept only syntax trees. On the syntax level, $\mathcal{M}$ provides mostly short to medium-length sentences with basic syntax. They include informal sentences (in the form of dialogs) but also simply-written formal sentences, so that frequent production rules from $\mathfrak{M}$ can establish a transition from informal to relatively formal utterances in the maintenance reports. Using a legacy corpus such as FTB (Abeillé et al., 2003) (originating from *Le Monde* articles) instead of $\mathcal{M}$ would be inadequate in our case, because of FTB's high syntactic complexity that is unlike the average syntax of $\mathcal{S}$ sentences.

4 The Lexical Level

In order to parse $\mathcal{S}$ efficiently we have preprocessed the text and extracted codes, abbreviations and equipment identifiers. We replaced these forms during parsing by a unique mark to avoid misinterpretations. We also detected misspelled/alternatively spelled words and replaced them by standard forms. As for $\mathcal{M}$, we removed sentences with an elliptic syntax (titles, sentences ending with ellipsis, etc.) using heuristic filters. After filtering we kept 48,693 sentences (650,847 words).

To evaluate $\mathcal{S}$'s vocabulary we have randomly chosen a subset $\mathcal{M}' \subset \mathcal{M}$, having the same volumetry as $\mathcal{S}$. Unsurprisingly, $\mathcal{S}$ has a significantly more restricted vocabulary than $\mathcal{M}'$: 5,505 different lexemes in the former vs. 9,374 in the latter. Their distribution is as follows:

	ADJ	NOUN	VERB	VN	PROPN
$\mathcal{S}$	7,137	**43,034**	12,616	9,295	**18,305**
$\mathcal{M}'$	**9,224**	36,610	**26,335**	**23,737**	8,358

where VN denotes past participles and PROPN proper nouns to which we added codes, abbreviations and equipment identifiers. We see that $\mathcal{S}$ has clearly more "proper nouns" and slightly more nouns than $\mathcal{M}'$, but all other parts of speech are underrepresented.

When words in $\mathcal{S}$ happen to be both frequent and complex, they occasionally undergo significant variation. Let us take the example of word "régénération," the sixth most frequent noun in $\mathcal{S}$ (485 occurrences in its standard form), which appears in the following alternative forms:

> régé: 1,117 times (apocope)
> Régé: 549 times (apocope)
> Rége: 14 times (apocope & accent error)
> rége: 12 times (apocope & accent error)
> rege: 11 times (unaccented apocope)
> Regé: 6 times (apocope & accent error)
> Rege: 5 times (unaccented apocope)
> régés: 5 times (apocoped plural)
> regé: 4 times (apocope & accent error)
> Regénération: twice (accent error)
> Régénaration, Régénèration, regenaration, regeneration, régeneration, régéneration: hapaxes (accent or spelling errors).

Variation is also frequent in English-origin words such as "bypass":

> bypass: 240 times
> by-pass: 147 times (with hyphen)
> Bypass: 19 times (capitalized)
> ByPass: twice (camel notation)
> By-pass, By-Pass, BY-pass: hapaxes.

Some abbreviation processes are peculiar such as the contraction "ppe" (for word "pompe") that occurs 117 times in the singular and 11 times in the plural number, or the apocope "échaff" (7 times) based on an erroneous ($\rightarrow$ two 'f's) spelling of word "échafaudage".

We encountered 920 cases of erroneous/non-standard spellings, involving 3,772 occurrences (out of which 588 were hapaxes).

The vocabulary is technical and has to remain unreduced. However, an interactive spell-checking and auto-completion device can be useful to avoid ambiguities, like in the cases of apocopes "aéro" or "régul" that can have a multitude of completions.

5 The Morphological Level

French is an uncased language, so that its morphological variation is focused mainly on conjugation for verbs and (less importantly) on number and gender of nouns and adjectives.

The use of verbs is very restricted in $\mathcal{S}$. While in $\mathcal{M}'$ we encountered 24 frequent different combinations of mode, tense, number and person, not counting infinitives and participles, in $\mathcal{S}$ there were only three frequent ones:

	P3s	P3p	F3s
$\mathcal{S}$	2,638	56	**106**
$\mathcal{M}'$	**3,524**	**159**	97

where P stands for present and F for future tense, 3 for third person and s/p for singular/plural. Episodic detection of other verb forms is often due to misspellings, such as in

> appel astreinte GN pour information que l'astreinte électrique ne peux rien faire de plus aujourd'hui !!!

where the P1s form of verb "pouvoir" is mistakenly used instead of the P3s form.

The low morphological variation of the $\mathcal{S}$ corpus comes as no surprise since maintenance reports use P3s and P3p to communicate the state of one or more devices at the time of report writing, and F3s (or F3p) for interventions that are scheduled in the near future, as in:

> la fin de la régénération de la chaîne 2 se terminera vers 7h45

6 The Syntax Level

Because of the conditions under which maintenance reports are authored, we notice a predominance of the "telegraphic style". This results in two phenomena: (1) elliptical language, as many obvious words are omitted for the sake of brevity; (2) chaotic syntax, where elementary rules of French sentence construction are broken. Typical examples are:

(1) fuite impulsion séparateur stable

which is a sentence containing neither verb nor determinant, consisting of three nouns and an adjective, and

(2) Faire avis sur fuite d'huile sulzer que si en augmentation voir consigne MPy

which seems like the (unpunctuated) transcription of an oral utterance. Here are completed version of these utterances, including implicit intentions, missing determinants and verbs:

(1′) *Nous avons constaté que la* fuite *de l'*impulsion *du* séparateur *est* stable.

(2′) *Il est conseillé de* faire *un* avis sur *la* fuite d'huile *du* sulzer *disant que si elle est en* augmentation *alors il faudra* voir *la* consigne *du* MPy.

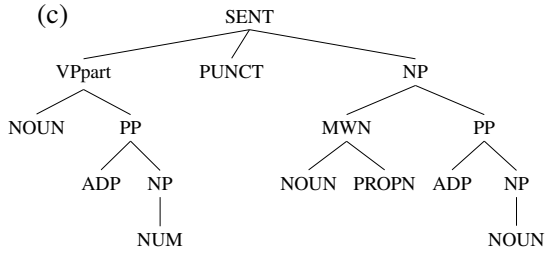
6.1 Parsing

We pre-processed the $\mathcal{S}$ corpus with regular expressions to replace all numerals and physical values with a single NUM tag and all device names and abbreviations with the PROPN tag: this has cut the number of distinct sentences in half, going from 30,851 sentences in the original corpus to 15,065. We then parsed the data using Stanford CoreNLP parser in order to obtain Phrase-Structure Grammar syntax trees of both corpora. We removed lexical leaves. We then extracted all production rules. Here is an example of this process:

(a) Arrêt à 20h00, arrêt TG1 à 20h15

(b) Arrêt à NUM , arrêt PROPN à NUM

(c)



(d) SENT → VPpart PUNCT NP, VPpart → NOUN PP, PP → ADP NP, NP → NUM, NP → MWN PP, MWN → NOUN PROPN, PP → ADP NP, NP → NOUN.

We calculated the occurrence frequency of every production rule in the corpus: it is the number of sentences in the syntax trees in which it is used (multiple use of a rule in the same syntax tree is not taken into account). The frequency of production rules follows a Zipf distribution: $\mathfrak{S}$ consists of 8,583 rules, the most frequent of which (namely PP → ADP NP) has a frequency of 18,584 and the distribution has a tail of 5,662 hapaxes (66% of the rules). On the other hand, $\mathfrak{M}$ consists of 30,930 rules, the most frequent (once again PP → ADP NP) having a frequency of 48,573 and the tail contains 23,203 hapaxes (75% of the rules). To give an example of the difference between $\mathfrak{S}$ and $\mathfrak{M}$, the SENT → PP rule which is typical of the telegraphic style and corresponds to expressions such as "en service" (frequency 263 in $\mathfrak{S}$) does not appear at all in $\mathfrak{M}$—on the other hand, the NP → DET NOUN rule that corresponds to the fundamental property of French nouns of being preceded by a determinant (a property that is often relaxed in telegraphic style) is the *second* most frequent rule in $\mathfrak{M}$ but only the *eleventh* in $\mathfrak{S}$.

6.2 Frequency-Based Subgrammars

Let T, N be fixed sets of terminals and non-terminals, and S an initial symbol. If R is a set of production rules we denote by $G(R)$ the corresponding formal grammar and by $L(R)$ the formal language recognized by $G(R)$. When $R \subset R'$ (while T, N, S remain fixed) then, obviously, $L(R) \subset L(R')$. Therefore by allowing a subset of production rules we obtain a sub-language. By keeping only the most frequent rules, we allow only for the most common syntactic features in the sub-language, and thereby the sub-language becomes a *simplified version* of the original language (Saggion, 2017, Ch. 4). Keeping a strongly reduced set of rules allows efficient manual definition of semantic rules, according to the principle of compositionality (Partee, 1995; Bird et al., 2019). The more production rules we allow, the more cumbersome it is to define the corresponding semantic rules. It is impossible to define semantic rules for an entire natural language, but it is possible to do so for controlled languages, provided their set of production rules is of reasonable size.

Input : Sentence $(w_1, \ldots, w_n)$, rules R
Output : Segments $(s_1, \ldots, s_m)$ where
$s_i := (w_{\ell(i)}, \ldots, w_{r(i)})$ for
$1 \leq i \leq n$, and
rest $:= (w_{n-j}, \ldots, w_n)$ for $j \geq 0$,
or $\emptyset$

if $(w_1, \ldots, w_n) \in L(R)$ **then**
  $s_1 := (w_1, \ldots, w_n)$;
  return $((s_1), \emptyset)$
end
else if $\nexists i, 1 \leq i < n$ *such that* $(w_1, \ldots, w_i) \in L(R)$ **then**
  return $(\emptyset, (w_1, \ldots, w_n))$
end
else
  $k \leftarrow 1$;
  $\ell(k) \leftarrow 1$;
  while $\exists r(k), \ell(k) \leq r(k) \leq n$ *such that* $(w_{\ell(k)}, \ldots, w_{r(k)}) \in L(R)$ **do**
    $s_k \leftarrow (w_{\ell(k)}, \ldots, w_{r(k)})$;
    $\ell(k+1) \leftarrow r(k) + 1$;
    $k \leftarrow k + 1$;
  end
  $k \leftarrow k - 1$;
  if $r(k) = n$ **then**
    return $((s_1, \ldots, s_k), \emptyset)$
  end
  else
    return $((s_1, \ldots, s_k), (w_{r(k)+1}, \ldots, w_n))$
  end
end

Algorithm 1: Left-Right Maximal Segmentation Algorithm

In our case, production rules are ordered by decreasing frequency and we can consider sets such as $\mathfrak{M}_{\geq 50}$: "the language produced by the terminals and non-terminals of the $\mathcal{M}$ corpus as well as the set of rules of frequency greater or equal to 50"; or $\mathfrak{S}_{\geq 2}$: "the language produced by the terminals and non-terminals of the $\mathcal{S}$ corpus using rules that are not hapaxe"s; etc.

We will use these sets as a base for tailoring controlled languages with a variable trade-off between formality and naturalness.

6.3 Left-Right Maximal Segmentation

According to (Angelov and Měchura, 2018), editors for controlled languages are

> of roughly two types. The first is the so called syntax editors which let the user manipulate a logical structure, while the actual text is just a byproduct. [...] The second kind is called predictive editors, which opt to work directly on the text level and guide the user by showing the set of possible continuations.

In the context of our project, both editor types are doomed to fail: boiler maintenance technicians under strong stress are probably not keen on visualizing syntax trees of their utterances, and a predictive editor would be incompatible with the high speed (not to say, haste) of the authoring act. Indeed, a technician having important information about the status of the equipment to transmit should be able to do so without any interference, and if there is time for improvement this can only happen a posteriori, after the authoring act is completed.

So the question is: how can we train technicians into using the controlled language in a way that is acceptable under the circumstances? The least intruding way would be to have a simple color code indicating successful/unsuccessful parsing, but then we fall into the other extreme: no information is given to the user on how to improve eir utterances, which can result in frustration when possible corrections have to be guessed.

The intermediate solution we adopt is to display a segmentation of the utterance into parsed sentences and, potentially, an unparsed rest. The rationale of this solution (loosely based on the pumping lemma for context-free languages) is that if an utterance (and in particular, a long one) is not a sentence for the controlled language, then there is a high probability (see Table 1) that some contiguous subsegments of it are nevertheless recognized as sentences. Starting from the left we mark the largest part of the utterance that is a sentence and iteratively repeat this process for the rest of the utterance, until we have reached a maximum sequence of segments that are sentences for the controlled language (see Alg. 1).

This gives the technician an understanding on how the utterance is decomposed into sentences by the parser. If the entire utterance is recognized by the parser, the author can leave it as such, otherwise e can intervene to change the phrase structure and attempt validation anew. If some words remain unparsed after the last segment, the author can attempt to incorporate them into the last segment, or to add text to produce a complete sentence out of them.

The success of this "training process" will depend on the coverage of the grammar. In Table 1

Table 1: Results of Maximal Left-Right Segmentation of the $\mathcal{S}$ Corpus

$\mathfrak{M}$ \ $\mathfrak{S}$		≥ 2 (1,411 rules)	≥ 3 (762 r.)	≥ 5 (412 r.)	≥ 10 (163 r.)	≥ 50 (21 r.)	$\emptyset$ (0 r.)
		Coverage (sentences with at least one segment)					
≥ 5	(2,204 rules)	90.87%	80.86%	78.8%	74.95%	64.93%	37.94%
≥ 10	(1,391 rules)	88.68%	77.11%	74.67%	70.47%	58.29%	28.72%
≥ 50	(460 rules)	81.72%	67.71%	64.84%	59.95%	43.38%	13.89%
≥ 100	(272 rules)	75.88%	61.18%	58.43%	53.48%	34.48%	6.96%
≥ 500	(81 rules)	61.36%	44.06%	42.08%	36.8%	18.15%	5.14%
		Rest (ratio between ratio length and utterance length)					
≥ 5	(2,204 rules)	0.07	0.08	0.09	0.11	0.16	0.25
≥ 10	(1,391 rules)	0.1	0.12	0.14	0.16	0.2	0.31
≥ 50	(460 rules)	0.19	0.23	0.25	0.28	0.33	0.46
≥ 100	(272 rules)	0.25	0.29	0.32	0.35	0.38	0.48
≥ 500	(81 rules)	0.42	0.46	0.48	0.5	0.48	0.51
		Average segment size (in words)					
≥ 5	(2,204 rules)	5.34	5.97	6.1	6.3	6.52	7.51
≥ 10	(1,391 rules)	5.63	6.18	6.31	6.51	6.73	7.88
≥ 50	(460 rules)	6.19	6.56	6.74	6.97	7.13	8.81
≥ 100	(272 rules)	6.39	6.7	6.88	7.09	7.22	10.2
≥ 500	(81 rules)	7.04	7.24	7.32	7.41	7.18	10.7

we present results of maximal left-right segmentation to the $\mathcal{S}$ corpus. Lines represent the various sets of $\mathfrak{M}$ rules used to segment the corpus. Taking all non-hapax rules we obtain the best results, but we must deal with semantic rules for thousands of syntax rules—on the other hand, when using only rules of high frequency, coverage drops drastically but so does the number of rules. The foremost right column represents the case where only $\mathfrak{M}$ rules are used to define the controlled language, the other columns consider the case where some $\mathfrak{S}$ rules are also allowed. The worst result occurs on the 5th line when only 81 $\mathfrak{M}$ rules and no $\mathfrak{S}$ rules are used: these conditions result in a very strict controlled language and it is not surprising that only 5.14% of the sentences of the existing corpus provide a segment. We can call $\mathfrak{M}_{\geq 500}$ the "strict strategy," where one wishes a syntactically simple controlled language at all cost.

Another strategy is $\mathfrak{M}_{\geq 5}$ which represents a significant effort to keep the controlled language close to standard French language. It involves preparing semantic rules for 2,204 syntactic rules, which is a considerable task. Using this approach, if technicians keep on writing as they did in the $\mathcal{S}$ corpus, in 37.94% of cases they will get a segmentation of, in average, three fourths of the utterance, with segments of an average length of 7.51 words. This is the "$\mathfrak{M}$-only at all costs" strategy.

A third strategy is to allow for additional rules coming from $\mathfrak{S}$ (note that the number of $\mathfrak{S}$ rules in the table stands for rules not already included in $\mathfrak{M}$). By taking a small number of $\mathfrak{S}$ rules, for example 21 rules of frequency ≥ 50, coverage increases significantly: we reach 64.93% vs. 37.94% in the previous strategy.

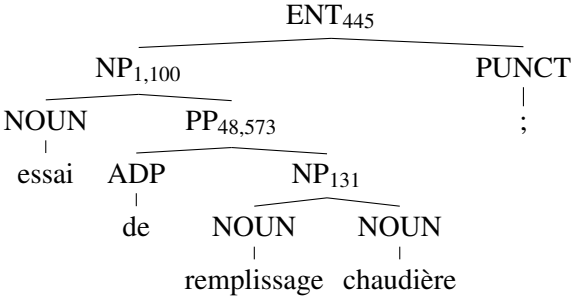
Finally the "most expensive" extreme is to take ≥ 2 rules from $\mathfrak{S}$ and ≥ 5 rules from $\mathfrak{M}$, which makes a total of 2,615 rules to manage, with a coverage of 90.87% of sentences and segments covering 93% of each sentence, in average. We consider this approach as a kind of overfitting, where one aims to reproduce the chaotic nature of the legacy language in a controlled environment, at a very high cost. Fortunately many intermediate solutions exist between these extremes.

In the following we will give examples of various utterances belonging or not to the controlled language for specific parameters.

7 Examples

7.1 $\mathfrak{M}_{\geq 50}$ and no $\mathfrak{S}$ rules

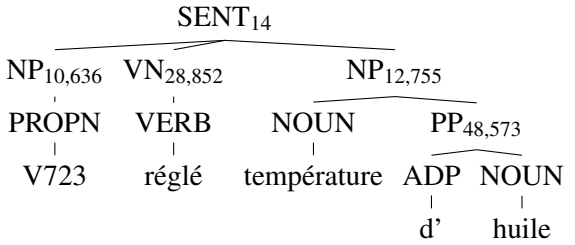
Our first example[3] is the one of a sentence that is successfully parsed with rule in $\mathfrak{M}_{\geq 50}$:



As we see the rule with lowest frequency is NP → NOUN NOUN, which can be found in $\mathcal{M}$ in expressions such as [[samedi]$_{\text{NOUN}}$ [après-midi]$_{\text{NOUN}}$]$_{\text{NP}}$.

7.2 $\mathfrak{M}_{\geq 10}$ and no $\mathfrak{S}$ rules

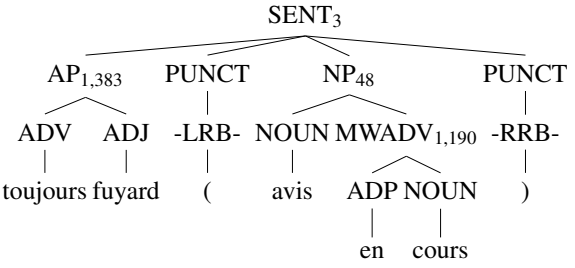
The following example uses $\mathfrak{M}$ rules of frequency ≥ 10 (with at least one rule of frequency ≤ 50) :



The least frequent rule here is SENT → NP VN NP, which appears only 14 times in the $\mathfrak{M}$ corpus, in syntagms such as [[Le coup de poing]$_{\text{NP}}$ [partit]$_{\text{VN}}$ [tout seul]$_{\text{NP}}$]$_{\text{S}}$

7.3 $\mathfrak{M}_{\geq 2}$ with no $\mathfrak{S}$ rules

The following example stretches syntax at its limits since it uses rare $\mathfrak{M}$ rules (of frequency higher than 2 but less than 10):

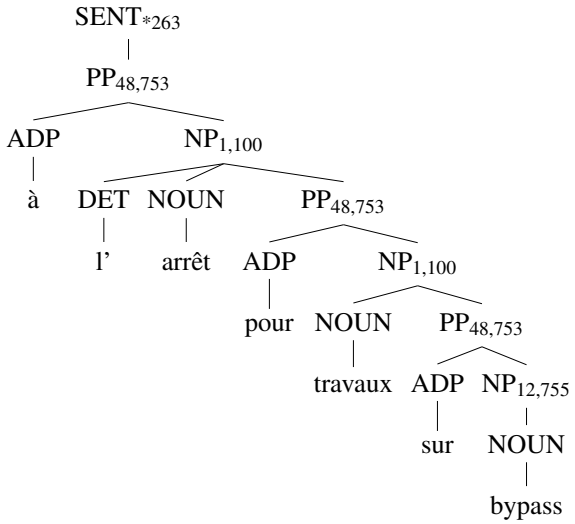


SENT → AP PUNCT NP PUNCT is a rare rule that appears almost by accident in a sentence such as [[Salut]$_{\text{AP}}$ [,]$_{\text{PUNCT}}$ [Marko]$_{\text{NP}}$ [?]$_{\text{PUNCT}}$]$_{\text{S}}$, which is hardly similar to our example. This sentence is clearly of telegraphic style. The rule NP → NOUN MWADV is not frequent either, it is attested in syntagms such as [[oui]$_{\text{NOUN}}$ [bien sûr]$_{\text{MWADV}}$]$_{\text{NP}}$.

[3] Indices in the syntax trees denote frequency in $\mathfrak{M}$, and starred indices denote frequency in $\mathfrak{S}$.

7.4 $\mathfrak{M}_{\geq 50} \cup \mathfrak{S}_{\geq 10}$

We now turn to an example that cannot be parsed entirely with $\mathfrak{M}$ rules and requires at least one $\mathfrak{S}$ rule, of frequency higher than 10:



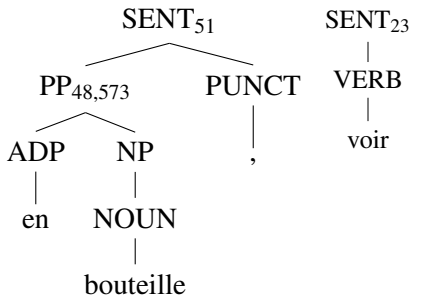
The rule from the $\mathcal{S}$ corpus is SENT → PP ("a sentence can be a prepositional phrase," which is typically telegraphic style) and it appears 263 times in $\mathcal{S}$. All other rules are quite frequent in $\mathfrak{M}$.

8 The Editor

To train users of the controlled language we have developed a device that parses word sequences on-the-fly, and displays maximal parsed segments using blue color (or some other graphical style) and brackets, from left to right. For example, for the utterance "en bouteille, voir schéma," the user will progressively see the following:

en
[en bouteille]
[en bouteille,]
[en bouteille,] [voir]
[en bouteille,] [voir] schéma

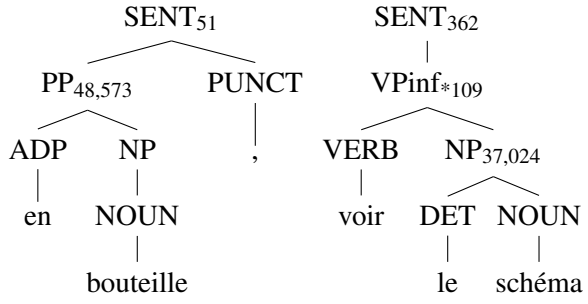
The last word cannot be absorbed by a segment if, e.g., only $\mathfrak{M}_{\geq 20} \cup \mathfrak{S}_{\geq 10}$ rules are allowed. This segmentation is based on the following two trees:



The second tree uses a rare rule SENT → VERB, which is of frequency 23 in $\mathfrak{M}$. At this point, the user can attempt a correction by adding a definite article "le" between "voir" and "schéma":

[en bouteille,] [voir le schéma]

The color changes to blue as the utterance is now entirely parsed, using the following trees:



The second tree uses the rule VPinf → VERB NP that belongs to $\mathfrak{S}$ with a frequency of 109.

We are planning to test the device and establish performance evaluation through user feedback.

9 Conclusion and Perspectives

We have presented a methodology for tailoring a controlled language out of the lexicon and morphology of a corpus, using the most frequent Phrase-Structure Grammar syntax rules of another corpus. We have applied this approach to a corpus of industrial equipment maintenance reports written in telegraphic style, as the former, and a corpus of Harlequin-like French novels as the latter. The goal is to make user utterances more easily interpretable. By allowing also some syntax rules from the original corpus, we obtain a better result in terms of coverage of utterances by syntax rules. By varying the number of allowed rules from the Harlequin-like novels and from the maintenance reports, we can change the formality/naturalness properties of the controlled language.

We have also presented a new editor type that is neither syntax-displaying nor predictive, but provides the user with information on the best possible left-to-right segmentation of the utterance into parsed sentences. This allows optional intervention by the user in order for the complete utterance to get parsed. The editor is purposely non-intrusive since the conditions under which maintenance reports are written do not always allow for a calm and reasoned reflection on syntax.

This is an ongoing project, the final goal of which is to achieve anomaly detection in reports, eventually correlating (timestamped) textual data with temporal series of data originating from sensors in the boilers, in search of anomalies. For this, formal interpretation of the reports can be useful but is not indispensable since text mining methods can compensate the lack of full interpretation. Another potential application is to correlate linguistic parameters of the corpus with author identities, since these are always provided in the reports. This would allow to evaluate the variability in lexicon, morphology and syntax due to author change.

10 Acknowledgments

This work has been realized in the frame of the European Regional Development Fund project AAP FEDER – LEARN IA, funded by the *Conseil régional de Bretagne* and the *European Union*.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In *Treebanks*, pages 165–187. Kluwer.

Krasimir Angelov and Michal Boleslav Měchura. 2018. Editing with search and exploration for controlled languages. In *Controlled Natural Language*, volume 304 of *Frontiers in Artificial Intelligence and Applications*, pages 1–10. IOS Press.

Flávia A. Barros, Neves Laís, Érica Hori, and Dante Torres. 2011. The ucsCNL: A controlled natural language for use case specifications. In *Proceedings of SEKE'2011, Miami Beach, Florida*, pages 250–253.

Steven Bird, Ewan Klein, and Edward Loper. 2019. *Natural Language Processing with Python*, 2nd edition. https://www.nltk.org/book/.

Peter Clark, William R. Murray, Phil Harrison, and John Thompson. 2010. Naturalness vs. Predictability: A key debate in controlled languages. In *CNL 2009 Workshop*, volume 5972 of *Springer LNAI*, pages 65–81.

Sébastien Ferré. 2012. SQUALL: A controlled natural language for querying and updating RDF graphs. In *CNL 2012*, volume 7427 of *Springer LNCS*, pages 11–25.

Norbert E. Fuchs. 2018. Understanding texts in Attempto controlled English. In *Proceedings of the 6th International Workshop on Controlled Natural Language (CNL 2018)*, volume 304 of *Frontiers in Artificial Intelligence and Applications*, pages 75–84. IOS Press.

Normunds Gruzitis, Peteris Paikens, and Guntis Barzdins. 2012. FrameNet resource grammar library for GF. In *CNL 2012*, volume 7427 of *Springer LNCS*, pages 121–137.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics*, 40:121–170.

Palmira Marrafa, Raquel Amaro, Nuno Freire, and Sara Mendes. 2012. Portuguese controlled language: Coping with ambiguity. In *CNL 2012*, volume 7427 of *Springer LNCS*, pages 152–166.

Barbara Partee. 1995. Lexical semantics and compositionality. In *An Invitation to Cognitive Science: Language*, volume 1, pages 311–360. MIT Press.

Jonathan Pool. 2006. Can controlled languages scale to the Web? In *CLAW 2006, AMTA 2006: 5th International Workshop on Controlled Language Applications*, pages 1–12.

Horacio Saggion. 2017. *Automatic Text Simplification*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool.

Approximating a Zulu GF concrete syntax with a neural network for natural language understanding

Laurette Marais
Voice Computing, NGEI, CSIR
laurette.p@gmail.com

Abstract

Multilingual Grammatical Framework (GF) domain grammars have been used in a variety of different applications, including question answering, where concrete syntaxes for parsing questions and generating answers are typically required for each supported language. In low-resourced settings, grammar engineering skills, appropriate knowledge of the use of supported languages in a domain, and appropriate domain data are scarce. This presents a challenge for developing domain specific concrete syntaxes for a GF application grammar, on the one hand, while on the other hand, machine learning techniques for performing question-answering are hampered by a lack of sufficient data. This paper presents a method for overcoming the two challenges of scarce or costly grammar engineering skills and lack of data for machine learning. A Zulu resource grammar is leveraged to create sufficient data to train a neural network that approximates a Zulu concrete syntax for parsing questions in a proof-of-concept question-answering system.

1 Introduction

In any Grammatical Framework (GF) domain grammar, typically called an application grammar, the abstract syntax defines the domain, in the sense that it provides the concepts of the domain and the ways in which they can be combined to construct meanings. The semantics is modelled, and hence restricted, by the abstract syntax, which leads each concrete syntax to be a controlled natural language (CNL) of a specific natural language, due to its limitation of semantics, syntax and vocabulary (Kuhn, 2014). The GF runtime enables both parsing of text strings into abstract trees, which is a kind of language understanding, and linearisation of abstract trees into text strings, which is a kind of language generation (Ranta et al., 2020).

GF-based CNLs have proven useful for developing language technology applications for low-resourced languages.[1] For example, Marais et al. (2020) presents a multilingual CNL used for speech-to-speech translation to enable healthcare providers to communicate with patients in their own language. Coverage of the application is restricted via a CNL in order to achieve high-quality speech translation in a high risk setting. In Marginean (2017), a GF domain grammar is used to perform question-answering (QA) in the medical domain.

The effort required to develop application grammars is reduced if a GF resource grammar (RG) exists for a language, since it can be used as a software library for the morphology and syntax of the language. Using an RG to develop an application grammar essentially involves mapping the semantic categories and functions in the application grammar to the syntactic categories and functions in the RG.

In this paper, we show how such a mapping can, in a sense, be learned by a neural network so that a workflow that relies heavily on grammar engineering skills can be replaced with one that requires machine learning skills. The two approaches are compared by considering the suitability of the resulting artifacts for the use case.

In Section 2 we present the application context for this work, namely a QA system where the concrete syntax we attempt to approximate is responsible for enabling language understanding. Then, in Section 3 we describe what a typical workflow for developing a Zulu concrete syntax would look like in an under-resourced development context. Section 4 presents the technique whereby the Zulu RG is leveraged alongside a domain abstract syntax to generate a dataset for training a neural network to

[1] For a recent audit of human language technology resources for Zulu, see Moors et al. (2018)

approximate a Zulu concrete syntax. A description of the neural networks trained and a comparison of them is given in Section 5. We discuss and contextualise the results in Section 6, before concluding with closing remarks in Section 7.

2 Overview of a grammar-based spoken QA system

The grammar-based system that provides the context for this work is a proof-of-concept spoken QA system that answers questions related to weather conditions for various locations in South Africa. The most important semantic concepts are place names, weather conditions, weather elements and time. The `WeatherFact` abstract syntax is centered around the notion of a weather fact, and for each supported language, two concrete syntaxes are included for expressing questions about weather facts and answers about weather facts, respectively. The OpenWeatherMap One Call API[2] is used to acquire up-to-date information with which to answer questions. It provides current weather information, historic weather data for 5 days, minute information for the next hour, hourly information for 48 hours and daily information for 7 days for any geographical coordinates. The semantics of this domain is by nature constrained, making a semantically limited CNL an appropriate choice for presenting a language interface to it.

The `WeatherFact` grammar allows questions about the weather for a fixed list of 33 locations in South Africa, including the biggest cities, provincial capitals, and cities and towns with airports. Furthermore, users can ask about the general weather conditions, or about specific weather metrics, such as cloud coverage or temperature, and time can be indicated by referring to a number (between 1 and 99) of minutes, hours or days in the past or future, or the concepts of yesterday and tomorrow. Typical English questions included in the grammar are 'What is the weather like in Johannesburg?' and 'What will the wind speed be in Cape Town in two days?'.

Figure 4 shows the basic architecture of the text-based part of the QA system. The `WeatherFact` multilingual GF grammar is responsible for parsing questions into semantic trees and linearising semantic trees into answers. Any given semantic tree can be linearised as a question or an answer, but parsing a question will result in an incomplete

[2]https://openweathermap.org/api/one-call-api

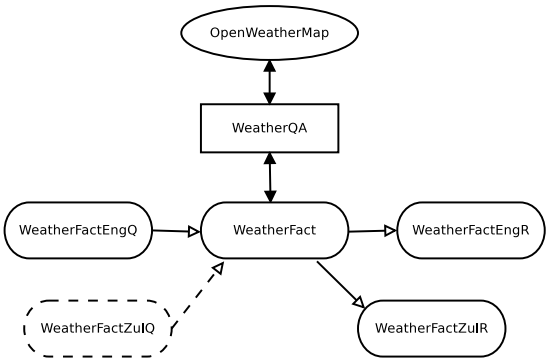


Figure 1: QA architecture

semantic tree – a missing sub-tree must be created and inserted by the QA system, so that the complete tree can be linearised as an answer. Figure 2 shows an example of a semantic tree, with the sub-tree that would be missing if it was created by parsing a question using the `WeatherFactEngQ` concrete syntax indicated in bold.

The incomplete semantic tree contains all the information necessary to inform the QA system of which query to send to the weather service, as well as how to extract the relevant information from the result in order to complete the tree. The root node informs the QA system what kind of information to expect in the tree. The OpenWeatherMap One Call API accepts geographical coordinates and returns a structured description of the current, past and future weather conditions as mentioned above. For example, in the case shown here, the QA system must use the `Location` information (in the form of a place name) in the tree to look up relevant geographical coordinates, which is used to send a query. The `TimeInstant` and `WeatherElement` information is then used to extract the appropriate information from the query result in order to supply the missing sub-tree.

3 Developing a question-parsing concrete syntax

For language generation in this context, it is entirely acceptable to provide a single way of expressing a certain meaning. However, a spoken QA system should allow some variation in how meaning is expressed by the user. Training data-driven models for this purpose typically requires large datasets of question-answer pairs (Bao et al., 2016), which are not available for many under-resourced languages, such as Zulu. Instead, a grammar-based language understanding system, developed for the relevant

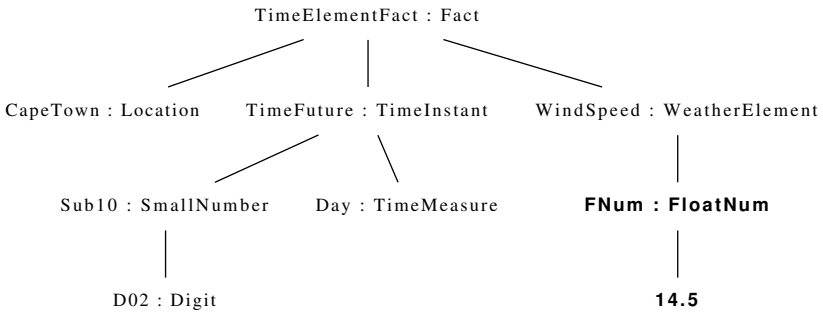


Figure 2: Semantic tree corresponding to the question 'What will the wind speed be in Cape Town in two days?' and the answer 'In two days, the wind speed in Cape Town will be 14.5 kilometers an hour.'

domain, remains a feasible approach to making spoken QA systems available for under-resourced languages. It then falls to the grammar engineer to produce a concrete syntax with an acceptable amount of flexibility for user input.

For Zulu, this is not usually as simple as bootstrapping from an existing concrete syntax for, for example, English, since the two languages are very dissimilar in terms of their linguistic characteristics and lexicon. For example, consider the English utterances 'Is it hot in Johannesburg?' and 'Is it windy in Johannesburg?', which concern the `WeatherElement` concepts of `Temperature` and `WindSpeed`, respectively. These utterances were translated by a Zulu language practitioner in order to develop an idiomatic Zulu concrete syntax, and the translations were given as *Kuyashisa eGoli?* and *Kunomoya eGoli?*. At first glance, the difference between the two Zulu utterances may appear to mirror that of the English utterances, but this is not the case.

Kuyashisa contains the verb root *-shis-*, which means to be hot, and it is used here in the present tense with the subject concord for noun class 17 (a locative class), to mean 'it is hot'. *Kunomoya*, on the other hand, contains the noun root *-moya*, meaning 'wind', and it is used in the present tense associative copulative construction, along with the subject concord for noun class 17, to mean 'it is with wind'. A concrete syntax developer would have to know that the two concepts are expressed using different syntactic constructions in Zulu, so that the Zulu linearisation category of `WeatherElement` in the application grammar reflects this variability. Ideally, the concrete syntax developer needs a tool to analyse the translated utterances to detect the syntactic and morphological characteristics in the domain data.

A Zulu resource grammar (RG) is currently under development[3], and in conjunction with the GF runtime and an appropriate lexicon, it provides the ability to parse Zulu utterances in order to inspect their linguistic structure. A typical workflow for developing a domain specific concrete syntax using the Zulu RG is as follows:

1. Develop the abstract syntax and an English concrete syntax.
2. Identify a representative subset of utterances from the grammar and render them in English.
3. Obtain a set of translations for the English utterances.
4. Analyse the syntactic constructions used in the Zulu translations (with the help of the RG), and extrapolate from them in order to implement the Zulu concrete syntax.
5. Generate Zulu utterances from the concrete syntax for review.

The resulting concrete syntax models the linguistic structures found in the translated domain data. The last step in the workflow requires specialised grammar engineering skills. In South Africa, where such skills are not taught at tertiary level, this workflow cannot be implemented widely. On the other hand, machine learning is widely taught and a growing community of natural language processing researchers in Africa (Orife et al., 2020) could exploit workflows that rely on data-driven techniques.

4 Data for domain specific language understanding

The obvious requirement for developing any data-driven solution is sufficient and appropriate data.

[3]https://github.com/LauretteM/gf-rgl-zul

Many efforts have been made and are currently underway to gather corpora for the under-resourced languages of Africa and South Africa (Barnard et al., 2014; Moors et al., 2018). In this work, we present a technique for *generating* a domain specific corpus. The use case admits of a semantically restricted CNL – in essence, we intend to develop a model for a Zulu CNL which is, in addition, implicitly restricted linguistically, rather than explicitly as with a GF concrete syntax.

4.1 What to learn?

A concrete syntax enables the GF parser to accept strings and to produce typed semantic tree structures, but this need not be the only options to consider as input and output for a neural network.

When considering Zulu text input, a first option is to use space separated tokens, but this presents some problems for machine learning. Zulu is an agglutinating language with a conjunctive orthography, resulting in tokens that often consist of long sequences of morphemes and which typically leads to sparsity in datasets. Alternation rules govern sound changes between adjacent morphemes, which makes the task of segmenting morpheme sequences non-trivial. However, Kotzé and Wolff (2015) have shown that segmenting on syllable boundaries, which are easily identified, can improve the performance of natural language translation systems. Furthermore, one could also consider character-level segmentation (Lee et al., 2017).

The Zulu RG provides additional options for input to a neural network, although it introduces a pre-processing step that may be subject to failure. Specifically, Zulu strings could be parsed by the GF runtime using the Zulu RG to produce syntax trees. Syntax trees, however, are hierarchical, and it may not be necessary to retain this structure in order to benefit from the pre-processing step. In fact, transformer sequence-to-sequence models use multi-head attention to learn the most relevant relationships between tokens in a sequence (Vaswani et al., 2017). In order to exploit existing transformer model implementations, the trees could be flattened into sequences of syntax function names. Additionally, the syntax trees could be mined for relevant information directly by, for example, extracting the lexical nodes to produce a lemma sequence.

A similar flattening might also be useful for the target of the neural network. Developing a concrete syntax that parses semantic function name sequences into semantic trees is trivial: each linearisation category is defined as a string, and each linearisation function is implemented to produce its own name followed by the strings of its children. The GF parser can then be exploited to restore the typed hierarchical structure.[4] Simpler sequences could be implemented by letting non-terminal functions produce an empty string followed by the strings contributed by its children. Such a concrete syntax essentially defines a natural language agnostic keyword question language that could be used by the QA system in exactly the same way as a concrete syntax for natural language questions.

4.2 What to learn from?

In the workflow discussed in Section 3, the representative subset of utterances from the domain grammar that is rendered in English in order to be translated, is chosen to be minimal and repetitive. For each semantic category, a template utterance is identified and the chosen category varied, so that its effect on the utterance can be seen. This kind of repetition is also a useful way to discourage a translator from introducing spurious variability in their translations that make it difficult to isolate the effects of the various semantic functions on the Zulu utterances. Variability, if required, is more effectively elicited by procuring utterances from different translators and by including variability in the source language.

For the `WeatherFact` grammar, 76 questions were generated via the English concrete syntax, which were translated by two language practitioners, resulting in 152 Zulu utterances. The English utterances included variations of utterances where it seemed natural in the English, such as 'How hot is it in Johannesburg?' for 'What is the temperature in Johannesburg?'. Although, Zulu numerals "are not a coherent morphosyntactic class" (Zerbian and Krifka, 2008), their behaviour is not domain specific information that requires elicitation from a translator, and hence the semantic functions for numbers were not varied in the elicitation data in the same way as other semantic functions.

The translations were parsed using the Zulu RG and a domain specific lexicon. Since the Zulu RG has not been completed yet, for the purpose of this experiment, it was extended, especially with re-

[4] Of course, this would only be possible if the function name sequence is defined by the concrete syntax. We discuss this caveat in Section 5.

gards to question words and their accompanying syntactic functions, in order to be able to achieve parses for 148 of the 152 translations, with at least one parsed Zulu utterance for each English utterance. The result is a set of 148 semantic-syntactic tree pairs.

4.2.1 Augmenting data within a CNL

The constraints on a domain of utterances imposed by the definition of an abstract syntax presents an opportunity to perform data augmentation that is likely to be semantically reliable. Augmentation can be done based on semantic categories, while the Zulu RG can be leveraged to produce grammatically correct Zulu utterances. If it is known that a certain change from one semantic tree to another is accompanied by a certain change from the corresponding syntax tree to another, this can be used to derive so-called augmentation rules. A rule would have the following form:

$T_A, T_B \rightarrow t_a, t_b$: Given a semantic tree T_1 and corresponding syntax tree t_1, if a semantic tree T_2 is acquired by substituting T_A in T_1 for T_B, and a syntax tree t_2 is acquired by substituting t_a in t_1 for t_b, then t_2 is the corresponding syntax tree of T_2.

A simple example from the `WeatherFact` domain is the following,

$$\{\text{Hour,Minute}\} \rightarrow \{\text{hora\_5\_6\_N,zuzu\_3\_4\_N}\}$$

which essentially states that whenever the noun stem *-hora* is used to express the meaning of `Hour`, the noun stem *-zuzu* can be used to express the meaning of `Minute` instead. With this rule, for example, the Zulu sentence *Bekushisa eMbombela emahoreni amathathu adlule?* ('How hot was it in Mbombela three hours ago?') gives rise to a new sentence, namely *Bekushisa eMbombela emizuzwini emithathu edlule?* ('How hot was it in Mbombela three minutes ago?'). Note the effect that the change in the class of the noun has on the modifying adjective *-thathu* ('three'), namely that *amathathu* becomes *emithathu*, as well as the relative clause based on the verb *-dlule* ('passed'), where *adlule* becomes *edlule*.

Rules are not limited with regards to the complexity of the sub-trees they contain. Figure 3 shows the rule which states that whenever the adverb *izolo*, which expresses `Yesterday`, is used, an adverbial phrase, which is linearised as *emahoreni amabili adlule*, can be used to express the notion of two hours ago (or `TimePast (Sub10 D02) Hour`).

A set of augmentation rules were developed with reference to the semantic-syntactic tree pairs. This was done by hand, but in principle, rules could also be derived from the semantic-syntactic tree pairs automatically, provided that the seed data is sufficiently representative. For example, we did not attempt to elicit examples of all numerals in the seed data in order to derive appropriate augmentation rules for numerals 1 to 99. Instead, these rules were defined by consulting linguistic texts, and could easily be reused for different domains. The augmentation rules were exhaustively applied to the seed data, which resulted in 341 254 unique semantic-syntactic tree pairs.

Next, the Zulu RG was used to linearise each syntax tree in order to obtain a Zulu utterance. Then, both the semantic and syntax trees were flattened and simplified to sequences of keywords and abstract lexical functions (effectively lemmas), respectively. The result was a dataset of 5-tuples (as shown in Table 1) that could be used to train sequence-to-sequence neural networks.

4.2.2 Review of augmentation algorithm

The semantic trees generated by the augmentation algorithm were compared to all those defined by the `WeatherFact` abstract syntax to determine if any semantic trees for questions were missing. The only trees not generated were those that contain a digit sequence starting with a zero. Given the semantics of the domain, this meant that all meaningful numbers (and therefore semantic trees) were generated.

A random sample of 100 5-tuples from the augmented data was selected and the Zulu linearisations, alongside their English equivalents, were presented to a language practitioner for review. The only errors discovered in the augmented data were the incorrect use of the immediate future and past tenses where the remote future and past tenses were required. This was deemed not to be a problem for this application, given the One Call API's time frame, for which the use of the remote tenses is unlikely. If it had presented a problem, the solution would have been to procure more translations that contain the remote tenses in Zulu, and to refine the augmentation rules accordingly.

{Yesterday,
TimePast (Sub10 D02) Hour} → {izolo_Adv,LocNPAdv (DetCN (DetQuant IndefArt NumPl)
(RelCN (AdjCN (PositA bili_A) (UseN hora_5_6_N))
(UseRCl TPerfTemp PPos (RelVP IdRP (UseV dlul_V)))))}

Figure 3: Example of an augmentation rule

Semantic tree	TimeElementFact Mbombela (TimePast (Sub10 D03) Minute) (Temperature ?)
Keywords	Mbombela TimePast D03 Minute Temperature
Syntax tree	PhrUtt NoPConj (UttS (UseCl TPerfPresTemp PPos (PredVP (UsePron (ProDrop it15_Pron)) (AdvVP (UseV shis_V) (LocNPAdv (AdvNP (DetCN (DetQuant IndefArt NumSg) (UseN Mbombela)) (LocNPAdv (DetCN (DetQuant IndefArt NumPl) (RelCN (AdjCN (PositA thathu_A) (UseN zuzu_3_4_N)) (UseRCl TPerfTemp PPos (RelVP IdRP (UseV dlul_V)))))))))))) NoVoc
Lemmas	it15_Pron shis_V Mbombela thathu_A zuzu_3_4_N dlul_V
Linearisation	bekushisa eMbombela emizuzwini emithathu edlule

Table 1: Example of a 5-tuple data point

4.2.3 Balancing the data

The presence of especially numbers in the grammar requires that the dataset be balanced to some extent. Since some kinds of utterances can contain any of the numbers below 100, these utterances far outnumber other kinds. The kind of utterances in this domain happens to correlate well with the length of the keyword sequence it is represented by, and so balancing was done on the length of the target keyword sequences by duplicating shorter utterances in the dataset. The final dataset contains 954 324 entries, and includes at least one example of every semantic question tree defined by the domain abstract syntax.

Usually, such duplication is not done when training machine learning systems, but in this case, we are aiming to approximate a concrete syntax. We will have achieved our goal if exactly those utterances which would have been modelled by a concrete grammar are handled correctly by the model, that is, those utterances that are in our implicit Zulu CNL. In effect, we are introducing drastic sample bias because our use case allows it.

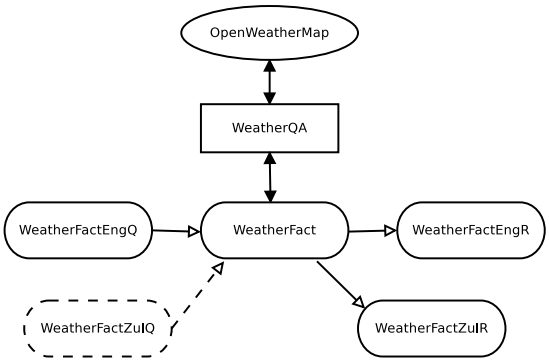


Figure 4: QA architecture

Model	F score	P score
token2key	1.00000	100%
syllable2key	1.00000	100%
char2key	0.99537	97.84%
syntax2key	1.00000	100%
lemma2key	0.99966	99.67%

Table 2: Comparison of different transformer models on a test set

5 Sequence-to-sequence models for language understanding

The PyTorch[5] implementation of a transformer sequence-to-sequence model of Vaswani et al. (2017) was used as the basis for a number of neural networks with varying input types, namely Zulu text strings, syntax function sequences and lemma sequences. For input based on the Zulu text strings, tokenisation was done on white space, syllable boundaries, and characters. Keyword sequences were used as the target. In total, five different models were trained on the augmented data, which was split into training, validation and test sets. Training was done over 3 epochs, which turned out to be sufficient for the models to converge. Table 2 lists the results obtained.

[5]https://pytorch.org

F score was calculated using the NLTK[6] implementation of ChrF score (with the default parameter values) on each respective test set. We have included the percentage of perfect predictions, which we have called the P score, because this is an indication of the number of times the QA system will be able to provide the user with exactly the information that was asked for, with no need to engage the user further.

The perfect or near perfect F for all the models show that they all succeed in learning to translate the controlled Zulu language to keyword sequences. Furthermore, the perfect or near perfect P scores mean that almost all output from the models can be parsed successfully using the keyword sequence concrete syntax. As such, the models could be used, in conjunction with a keyword concrete syntax, to approximate a Zulu concrete syntax for language understanding.

5.1 Bias and generalisation

We have knowingly introduced sample bias into our models by training exclusively on synthetic data. The next step would be to try to determine if any of the models generalise beyond the implicit Zulu CNL represented by the training data to a more semantically constrained CNL. In other words, can the model(s) accurately understand independently sourced Zulu utterances that express meanings defined in the abstract syntax?

To investigate this, the original set of English seed utterances were translated by two new Zulu translators. The new translations only contain a 13% overlap with the original translations, which seems to confirm the need for allowing variability in user input for a Zulu QA system. As before, the translations were parsed using the Zulu RG and in this case, we did not extend the grammar to parse the new translations. However, in order to compare the different models, we only used those new translations that could be parsed using the RG as is: we used 71 utterances, and discarded 81. As the RG becomes more complete, the former number should rise and the latter should drop. For now, we note that this step introduces some bias in the evaluation towards the syntactic structures currently covered by the RG.

The seed data was designed to be minimal and repetitive, as noted earlier, which makes it an unsuitable evaluation set. In order to achieve a more balanced evaluation set, we augmented the 71 parsed utterances (resulting in 138 896 utterances) and sampled it so that the final evaluation set contained 100 independently generated 5-tuples, balanced according to keyword sequence length. It should be noted that this augmentation step is only possible on utterances that can be parsed using the RG.

[6]http://www.nltk.org/

Model	F score	P score
token2key	0.65185	33%
syllable2key	0.83098	**54%**
char2key	0.81897	52%
syntax2key	0.81894	52%
lemma2key	**0.83629**	53%
lemma2key*	0.18702	0%

Table 3: Comparison of different transformer models on independently generated data

The bias introduced by limiting our evaluation to such utterances becomes clear when inspecting the evaluation set: although all the weather elements included in the domain grammar appear in the set of newly parsed utterances, this is only true for the present tense. The syntactic constructions used in the new translations to express the notion of *temperature* in the past and future tenses are the only ones currently covered by the RG. As a result, the `Temperature` keyword occurs in 52 of the 100 keyword sequences.

From Table 3, we can see that the *lemma2key* model performs the best in terms of F score, with *syllable2key*, *syntax2key* and *syntax2key* achieving comparable scores. In fact, the outlier is the *token2key* model trained on space separated tokens, which seems to confirm that the orthography of Zulu presents a problem for machine learning, and that, presumably, any attempt to segment the text systematically produces an improvement.

For reference, we have included the F and P scores for a *lemma2key** model trained only on the original seed data. Table 4 gives a comparison of some examples in the evaluation set for each of the two lemma-based models. The predictions have been post-processed to include only the tokens appearing before the first end-of-sequence symbol. It is evident that the model trained on the seed data has simply learned to produce sequences that mostly consist of the token 'Johannesburg', hence its P score of 0%.

Target	Augmented model prediction
NorthernCape Yesterday	NorthernCape Yesterday
EasternCape WindSpeed	EasternCape Clouds
Durban Yesterday Temperature	Durban Temperature
Limpopo TimePast D01 Hour	Limpopo TimePast D01 Hour
Target	**Seed model prediction**
NorthernCape Yesterday	Johannesburg Johannesburg
EasternCape WindSpeed	Johannesburg Temperature
Durban Yesterday Temperature	Johannesburg
Limpopo TimePast D01 Hour	Johannesburg Johannesburg Johannesburg

Table 4: Comparing the *lemma2key* model trained on the augmented data and the seed data

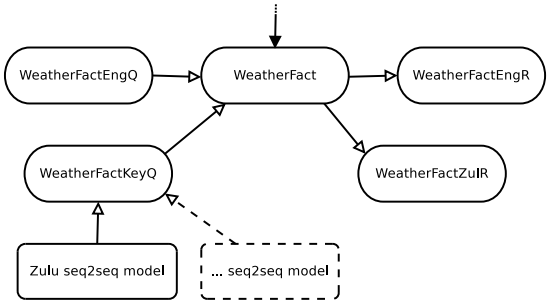


Figure 5: Grammar component of the QA architecture, adapted to include a Zulu sequence-to-sequence model for questions

6 Discussion

We have shown that the Zulu RG can be leveraged in conjunction with a domain abstract syntax to augment a very small set of manually translated data to a sufficiently large set of synthetic data, which essentially represents all the utterances a concrete syntax would be expected to cover. Using this dataset, a variety of different transformer models can be trained and incorporated into a pipeline that would perform almost exactly like the GF runtime enabled by a Zulu concrete syntax, with regards to parsing natural language into typed semantic tree structures. From this we conclude that it is possible to approximate a Zulu GF concrete syntax with a neural network to perform language understanding. Figure 5 shows the adapted configuration.

The language that is correctly "understood" by the models (i.e. the set of Zulu strings that lead to perfect predictions) has also been shown to be somewhat larger than what a concrete syntax might include. While it does not make sense to talk about the language accepted by a neural model, the model is integrated into the QA system in conjunction with a language agnostic keyword concrete syntax. Hence, those Zulu sentences that cause the neural model to produce strings that are in the keyword sequence language can be thought of as being "accepted". And while it is not possible to know exactly which Zulu sentences are accepted in this way, it is known which keyword sequences can be reconstructed into typed trees defined by the abstract syntax. In the long definition of a CNL, with regards to the C in CNL, Kuhn (2014) proposes that a CNL must be restricted in terms of "lexicon, syntax and/*or* semantics" (our emphasis) and also that it be "explicitly and consciously defined". It is certainly the case that the domain abstract syntax, as well as the keyword concrete syntax, is explicitly and consciously restricted in terms of semantics, while the input to the system is natural Zulu. Hence, we contend that the conjunction of a neural model and a keyword concrete syntax as described here does indeed implement a controlled natural language.

This work differs in some important ways from other text augmentation attempts. The goal of text augmentation is usually to *improve* machine learning by supplementing existing data (Duwairi R, 2021). The seed data is typically enough to train a reasonable system, but significant improvements can be made via augmentation. In this work, due to the absence of suitable data, seed data was generated via manual translation. This is an expensive way of obtaining data, and so the goal was to start with a minimal seed corpus, which we showed to be woefully insufficient to train a useful model.

Although the effective gain in data size for text augmentation techniques is not often reported, with authors instead reporting on improvements in system performance (Sharifirad et al., 2018; Kobayashi, 2018; Rizos et al., 2019; Şahin and Steedman, 2018), an increase of 5 times the original dataset is reported by Wang and Yang (2015),

while Duwairi R (2021) report a tenfold increase. This is in sharp contrast to the more than 2000-fold increase achieved here, from 148 to 341 254 unique utterances.

Furthermore, text augmentation has often focused on text classification tasks (Sharifirad et al., 2018; Kobayashi, 2018; Rizos et al., 2019), as opposed to sequence generating tasks, such as POS-tagging (Şahin and Steedman, 2018). Our work is similar to the latter in that a sequence of labels is generated, but, to the best of our knowledge, our work differs from any previous work with regards to the novelty of the target sequences generated by the augmentation process. In classification tasks, augmentation techniques have been aimed at producing more examples that preserve (or at most flip) the labels of the existing data (such techniques are "meaning preserving" (Rizos et al., 2019) in different senses, depending on the task), while the work of Şahin and Steedman (2018) either reduces or rearranges POS tags of existing target sequences (along with their corresponding source sequences) by cropping and rotating Universal Dependency trees.

Our augmentation technique, in contrast, leverages a domain grammar that models the semantics of a domain to produce entirely new target sequences. In addition, it leverages a Zulu resource grammar that models the linguistics of the natural language to produce corresponding source sequences. A useful connection has been explored between Universal Dependency trees and the GF RG library (Kolachina and Ranta, 2019), and in this sense also, our work is most similar to that of Şahin and Steedman (2018). However, the addition of a semantic domain abstract syntax has been the key to generating pairs of new source and target sequences for the task of language understanding. For example, substituting the notion of 'yesterday' with that of 'two hours ago', as per the rule in Figure 3, produces a new data point where the source sequence (via the syntax tree) *and* the target sequence (via the semantic tree) contain new tokens.

The domain abstract syntax increases the factor by which data can be augmented, while also imposing limitations on the structure of new data points that are generated. Future work will include experimenting with the complexity of domain grammars to better understand the ability of the augmentation technique to scale to larger and more complex domains, as well as to study the ability of neural networks to deal with increasingly complex constructs.[7]

7 Conclusion

The workflow described in this paper does not require any grammar engineering skills. Instead, it relies on the ability to use the Zulu RG and GF runtime to parse Zulu text and linearise syntax trees. This is a significant advantage in contexts where grammar engineering skills, especially in conjunction with knowledge of Zulu and its use in any given domain, is costly or scarce.

In addition to approximating a concrete syntax, we have shown that certain transformer sequence-to-sequence models, trained on synthetic augmented data, have some ability to generalise beyond the linguistic structures found in the seed data. This is an improvement on the use of a concrete syntax, especially since the ability to deal with variability in user input is important in spoken QA systems. An attempt was made to evaluate the extent of this kind of generalisation, although a more accurate assessment will only be possible once the Zulu RG is complete, since it forms the basis for generating a balanced evaluation set from seed data.

The neural networks developed in this work are unidirectional, as opposed to a concrete syntax which enables the GF runtime in both the parsing and linearising directions. Future work will include training neural networks in the opposite direction, namely to generate Zulu utterances from semantic trees. Since the language generation aspect of the QA system only requires one way of expressing meaning, it is expected that the technique presented here would achieve similar success.

The data augmentation step, which centres on a GF RG for the language, forms the core of the technique. In principle, therefore, it could be applied to any under-resourced language for which a GF RG exists.

Acknowledgements

The author would like to thank three anonymous reviewers for their insightful comments towards improving this manuscript.

[7] Thanks to an anonymous reviewer for this suggestion.

References

Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2503–2514.

Etienne Barnard, Marelie H Davel, Charl van Heerden, Febe De Wet, and Jaco Badenhorst. 2014. The NCHLT speech corpus of the South African languages. Workshop Spoken Language Technologies for Under-resourced Languages (SLTU).

Abushaqra F. Duwairi R. 2021. Syntactic- and morphology-based text augmentation framework for arabic sentiment analysis. *PeerJ Computer Science*, 7.

Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 452–457, New Orleans, Louisiana. Association for Computational Linguistics.

Prasanth Kolachina and Aarne Ranta. 2019. Bootstrapping UD treebanks for delexicalized parsing. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 15–24, Turku, Finland. Linköping University Electronic Press.

Gideon Kotzé and Friedel Wolff. 2015. Syllabification and parameter optimisation in Zulu to English machine translation. *South African Computer Journal*, (57).

Tobias Kuhn. 2014. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully Character-Level Neural Machine Translation without Explicit Segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378.

Laurette Marais, Johannes A. Louw, Jaco Badenhorst, Karen Calteaux, Ilana Wilken, Nina van Niekerk, and Glenn Stein. 2020. AwezaMed: A Multilingual, Multimodal Speech-To-Speech Translation Application for Maternal Health Care. In *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*, pages 1–8.

Anca Marginean. 2017. Question answering over biomedical linked data with grammatical framework. *Semantic Web*, 8(4):565–580.

Carmen Moors, Ilana Wilken, Karen Calteaux, and Tebogo Gumede. 2018. Human language technology audit 2018: Analysing the development trends in resource availability in all south african languages. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '18, page 296–304, New York, NY, USA. Association for Computing Machinery.

Iroro Orife, Julia Kreutzer, Blessing Sibanda, Daniel Whitenack, Kathleen Siminyu, Laura Martinus, Jamiil Toure Ali, Jade Abbott, Vukosi Marivate, Salomon Kabongo, Musie Meressa, Espoir Murhabazi, Orevaoghene Ahia, Elan van Biljon, Arshath Ramkilowan, Adewale Akinfaderin, Alp Öktem, Wole Akin, Ghollah Kioko, Kevin Degila, Herman Kamper, Bonaventure Dossou, Chris Emezue, Kelechi Ogueji, and Abdallah Bashir. 2020. Masakhane – Machine Translation For Africa.

Aarne Ranta, Krasimir Angelov, Normunds Gruzitis, and Prasanth Kolachina. 2020. Abstract Syntax as Interlingua: Scaling Up the Grammatical Framework from Controlled Languages to Robust Pipelines. *Computational Linguistics*, 46(2):425–486.

Georgios Rizos, Konstantin Hemker, and Björn Schuller. 2019. Augment to prevent: Short-text data augmentation in deep learning for hate-speech classification. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 991–1000, New York, NY, USA. Association for Computing Machinery.

Gözde Gül Şahin and Mark Steedman. 2018. Data augmentation via dependency tree morphing for low-resource languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5004–5009, Brussels, Belgium. Association for Computational Linguistics.

Sima Sharifirad, Borna Jafarpour, and Stan Matwin. 2018. Boosting text classification performance on sexist tweets by text augmentation and text generation using a combination of knowledge graphs. In *Proceedings of the 2nd workshop on abusive language online (ALW2)*, pages 107–114.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR*, abs/1706.03762.

William Yang Wang and Diyi Yang. 2015. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using #petpeeve tweets. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2557–2563, Lisbon, Portugal. Association for Computational Linguistics.

Sabine Zerbian and Manfred Krifka. 2008. Quantification across bantu languages. *Quantification: A cross-linguistic perspective*, 64:383–414.

The Grammar of PENG^{ASP} Explained

Rolf Schwitter
Department of Computing
Macquarie University
Sydney, NSW, 2109, Australia
Rolf.Schwitter@mq.edu.au

Abstract

In this paper we present the controlled language and the grammar of the PENG^{ASP} system and explain how the new version of the grammar has been implemented in a logic programming framework. The grammar is now bi-directional and can be used to translate a specification written in controlled language into an executable answer set program and vice versa. The grammar is highly configurable for different application scenarios and can be used for incremental text processing together with a predictive authoring tool.

1 Introduction

A controlled language is a restricted version of a natural language which has been engineered by reducing the complexity of its grammar and/or its vocabulary to meet a particular purpose (Schwitter, 2010; Kittredge, 2003). Human-oriented controlled languages aim to improve the communication between humans or the readability of technical documentation for humans who are often not native speakers of the language. Machine-oriented controlled languages aim to improve machine translation of (technical) documentation or to support automatic reasoning via the translation of the language into a knowledge representation language.

PENG^{ASP} (Guy and Schwitter, 2017) is a machine-oriented controlled language and is in this respect similar to Attempto Controlled English (Fuchs et al., 2008), Computer-Processable Language (Clark et al., 2005), and Logical English (Kowalski, 2020). However, in contrast to these other three controlled languages, specifications written in PENG^{ASP} are exclusively translated into executable answer set programs (Gelfond and Kahl, 2014; Gelfond and Lifschitz, 1988). Answer set programming offers a rich declarative knowledge representation language for non-monotonic reasoning and is supported by high performance reasoning tools (Gebser et al., 2019). In contrast to the grammar of the PENG^{ASP} system introduced in Guy and Schwitter (2017), the latest version of the grammar is now bi-directional. This means that the same grammar can be used for processing a specification and for verbalising an answer set program.

According to the PENS scheme (Kuhn, 2014), controlled languages can be classified along four dimensions: precision (P), expressiveness (E), naturalness (N), and simplicity (S). Each of these dimensions is then measured on a scale of 1 to 5. In addition to these four dimensions, nine properties are used to identify the type of a controlled language. Following this scheme, the controlled language PENG^{ASP} can be classified as P^{5}E^{3}N^{4}S^3 A,W,F. This means PENG^{ASP} is a language with fixed semantics (P^5); offers medium expressive power (E^3); uses natural sentences (N^4); and requires a description of more than 10 pages (S^3). Furthermore, PENG^{ASP} originated from academia (A), is intended to be written (W), and to be formally (F) represented as an answer set program. In other words, PENG^{ASP} is a high-level specification language for answer set programs that combines the readability and understandability of natural language with the precision and expressiveness of a declarative knowledge representation language.

The rest of this paper is structured as follows: In Section 2, we outline the requirements to the grammar of PENG^{ASP}. In Section 3, we give a brief introduction to answer set programming, since PENG^{ASP} is closely related to this formal target language. In Section 4, we introduce a motivating example that illustrates some features of the language and show how the corresponding answer set program looks like. In Section 5, we reveal more details about the design of the controlled language with a particular focus on the usage of certain grammatical constructions. In Section 6, we take a look at the implementation of the bi-directional grammar; and in Section 7, we conclude.

2 Requirements to PENG^{ASP}

The controlled language PENG^{ASP} and its grammar have been designed with a number of requirements in mind. Firstly, a controlled language specification should be translatable into an executable answer set program. Secondly, the grammar for this language should be highly configurable for different application scenarios, also for scenarios that do not necessarily require the full power of answer set programming. Thirdly, the same grammar should support the processing of a specification and the verbalisation of an answer set program; therefore, the grammar should be bi-directional. Fourthly, the grammar should also be useful to support the writing process of a specification in an incremental way, in particular with respect to generating lookahead information and resolving anaphoric expressions.

A possible processing strategy is to translate a controlled language specification into a syntax tree and then transform this tree into an answer set program. A better strategy is to generate the answer set program directly during the parsing process and design the grammar in such a way that it can serve as a language processor as well as a language generator. To achieve this, we start from a definite clause grammar and specify the grammar rules for the controlled language PENG^{ASP} in this unification-based notation (Pereira and Warren, 1980), and then use SWI Prolog (Wielemaker et al., 2012) as programming language. However, when Prolog is directly used to evaluate a definite clause grammar in a system like ours that heavily relies on incremental processing and user interaction, then Prolog's backtracking search strategy is not optimal, since it forgets all the previous work that it has done after a user interaction. To solve this problem, we use a chart parser in conjunction with the definite clause grammar to get the effect of a more complete parsing strategy that remembers substructures that it has already parsed (Gazdar and Mellish, 1989). For the processing of a specification, we transform the definite clause grammar into an alternative notation using a logic programming technique called term expansion (Wielemaker et al., 2012). The resulting notation is easier to process by a chart parser and the chart can then be used to extract the information that is required to support the writing process on the user interface level. For the verbalisation of an existing answer set program, the definite clause grammar can directly be used and does not need to be transformed into another format for chart parsing, since verbalisation does not require any user interaction in our system.

Bi-directionality is a key feature of our grammar and distinguishes the PENG^{ASP} system from other controlled language processors (Fuchs et al., 2008; Clark et al., 2005). Bi-directionality requires that we can (a) feed a specification *S* as input to the grammar *G* and get an answer set program *A* as output, and (b) feed the answer set program *A'* as input to the same grammar *G* and get a semantically equivalent version *S'* of the original specification as output. Figure 1 illustrates this form of lossless semantic round-tripping (Schwitter, 2020).

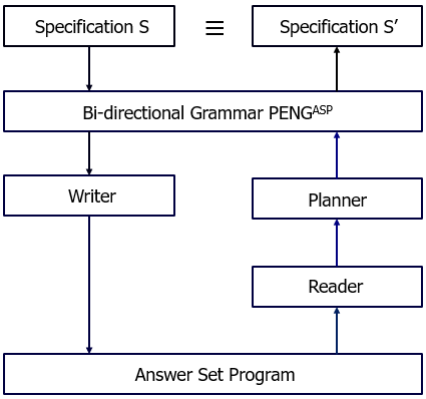


Figure 1: Round-tripping in PENG^{ASP}

In order to achieve semantic round-tripping, we use a Writer module that converts the internal version of the answer set program built by the grammar into an executable answer set program. In the case of verbalisation, a Reader module is used to read the executable answer set program and to produce a linguistically processable version of that answer set program. Since this processable version may contain certain redundancies, a Planner module is used that applies micro-planning tactics to aggregate redundant information such as subject aggregation and to deal with the identification of definite descriptions. The output of the Planner is a more compact version of the answer set program that is sent to the grammar and used to verbalise the answer set program.

3 Answer Set Programming

Answer set programming (ASP) is a declarative programming paradigm for knowledge representation and reasoning (Gelfond and Kahl, 2014; Gelfond and Lifschitz, 1988). ASP has been developed in the field of logic programming and non-monotonic reasoning and has been applied to a

wide range of areas in artificial intelligence (Erdem et al., 2016). ASP is supported by powerful reasoning tools and offers a rich representation language that allows for recursive definitions, negation, constraints, aggregates, optimization statements, and external functions (Gebser et al., 2019). An ASP program consists of a set of rules of the following form:

h_1 ; ... ; h_m :- b_1 , ... , b_n .

Here each h_i is a classical atom and b_i is a literal for $m \geq 0$ and $n \geq 0$. A classical atom h_i is either a positive atom of the form $p(t_1,\ldots,t_k)$ or its strong negation of the form $-p(t_1,\ldots,t_k)$, where p is a predicate name, $t_1,\ldots,t_k$ are terms, and $k \geq 0$ is the arity of the predicate name. A literal b_i is of the form *A* or not *A*, where *A* is a classical atom or an atom over a built-in comparison predicate used to compare terms, and the connective not denotes weak negation (aka negation as failure or default negation). Note that a literal of the form not *A* is assumed to hold unless the atom *A* is derived to be true. In contrast, strong negation of an atom holds only if it can be derived. The if-connective ':-' separates the head of a rule from its body. Intuitively, if all positive literals in the body of a rule are true and all negative literals are satisfied, then the head of the rule must be true. The connective ';' denotes a disjunctive head. A disjunctive head holds if at least one of its atoms is true. An ASP rule with an empty body (and without the if-connective) is called a fact, and an ASP rule with an empty head (but with the if-connective) is called an integrity (strong) constraint.

An extension of ASP that is relevant for our work are choice rules. A choice rule has the form:

l{e_1 ; ... ; e_m}u :- b_1 , ... , b_n .

Here e_i is a choice element of the form $a:L_1,\ldots,L_k$, where a is a classical atom, L_i are literals, and l and u are integers which express lower and upper bounds on the cardinality of elements. Intuitively, a choice rule means that if the body of the rule is true, then an arbitrary number of elements can be chosen as true as long as this number complies with the upper and lower bounds.

Note also that the input language to the ASP tool *clingo* (Gebser et al., 2019) supports double default negated literals in strong constraints. Furthermore, the language also supports weak constraints. In contrast to strong constraints, weak constraints do not eliminate answer sets but weight and prioritise them; more about this later.

4 A Motivating Example

The grammar of the PENG^{ASP} system translates a specification written in controlled language into an executable ASP program. The following example is an excerpt of a specification that contains information about students and their enrolments.

1. COMP3160 and COMP3220 are units.
2. Liam is a student and Olivia is a student.
3. Every student is either enrolled in COMP3160 or is enrolled in COMP3220.
4. If a student withdraws from a unit then the student is not enrolled in that unit.
5. It is not the case that Liam is enrolled in COMP3220.
6. Olivia withdraws from COMP3160.
7. Who is enrolled in COMP3220?

This specification consists of class assertions in (1) and (2); two conditional statements in (3) and (4); a constraint in (5), an unconditional statement in (6), and a wh-question in (7).

Listing 1: Answer Set Program

```
named(1, comp3160). class(1, unit).
named(2, comp3220). class(2, unit).
named(3, liam). class(3, student).
named(4, olivia). class(4, student).
1 { prop(X, 1, enrolled_in) ;
    prop(X, 2, enrolled_in) } 1  :-
  class(X, student).
-prop(X, Y, enrolled_in) :-
  class(X, student),
  pred(X, Y, withdraw_from),
  class(Y, unit).
:- prop(3, 2, enrolled_in).
pred(4, 1, withdraw_from).
answer(PN) :-
  named(X, PN), prop(X, 2, enrolled_in).
```

As we can see in Listing 1, the translation of the class assertions in (1) and (2) results in a number of facts in the ASP program. The two conditional statements are translated into two ASP rules. The first one (3) is translated into a choice rule that implements an exclusive disjunction describing alternative ways to form answer sets. The second one (4) is translated into a rule that contains a strongly negated atom as head. This rule eliminates answer set solutions if the body of the rule is true. The constraint in (5) results in a strong constraint in ASP and weeds out a particular solution from the generated answer sets. The statement in (6) is translated into a fact and the wh-question in (7) into an ASP rule with a specific atom (answer/1) in its head.

The ASP program uses a reified notation with a small number of predefined predicate atoms (e.g., `class/2`, `named/2`, `pred/3`, and `prop/3`). These predicate atoms can take variables, constants, positive numbers, or functional terms as arguments. Constants represent content words that occur in a specification and positive numbers replace existentially quantified variables in the program.

5 The Language: PENG^{ASP}

PENG^{ASP} can be used to make statements, enforce constraints, ask questions about a specification, and issue directives. There is not enough space in this section to cover all grammatical constructions of the language; therefore, we focus on the most important ones and provide selected examples. Syntactically, PENG^{ASP} distinguishes between simple and composite sentences. Each sentence consists of one or more clauses, and each clause has one main verb. Within a composite sentence, clauses may be joined via coordination or subordination, thus forming a compound or a complex sentence, respectively. The tense of the verb in a clause is either simple present, present continuous or future continuous depending on how the clause is used.

It is useful to introduce the concept of a core clause that forms the building block for simple and composite sentences in PENG^{ASP}. A core clause has the following canonical structure:

```
subject + predicator + (complements)
```

The subject and the predicator are always mandatory in a core clause. The subject is realised by a noun phrase and the predicator by a verb of a verb phrase. The selection of complements depends on the verb; dropping a complement either results in an incomplete core clause or a significant change in the meaning of the verb. A core clause forms the predicate-argument structure of a simple sentence. Adjuncts can follow the complement(s), but they are always optional, since they are not required to complete the meaning of a core clause. Phrases that occur in adjunct position serve exclusively as verbal modifiers; they add additional information like spatial or temporal information to the meaning of a core clause.

5.1 Making Statements

Simple statements can be made with the help of a simple sentence like (8) that is based on a core clause. The verb of this sentence can be modified for instance by a prepositional phrase that occurs in adjunct position; and this modification can be expressed as part of a simple sentence like (9). A positive clause like (9) can be rendered negative by the insertion of a strong negation (10).

8. Liam arrives.
9. Liam arrives at 09:00.
10. Liam does not arrive at 09:00.

Complex statements can be expressed with the help of compound sentences like (11), complex sentences like (12), verb phrase coordination like (13), and noun phrase coordination for class assertions like (14).

11. Liam studies at Macquarie University and Liam is enrolled in COMP3160.
12. Liam who studies at Macquarie University is enrolled in COMP3160.
13. Liam studies at Macquarie University and is enrolled in COMP3160.
14. Liam, Olivia, and Rona are students.

The compound sentence (11) uses two independent clauses that are joined by a coordinating conjunction (*and*). The complex sentence (12) consists of an independent clause and a dependent clause in the form of an embedded relative clause that modifies the proper name with the help of a subordinating conjunction (*who*). In (13) verb phrase coordination shares the same subject and in (14) noun phrase coordination (enumeration) shares the same complement. Note that the sentences (11-13) are syntactic variations of each other and result in the same ASP representation.

5.2 Making Conditional Statements

Like simple statements, simple conditional statements can also be expressed with a simple sentence that is based on a core clause. But in this case, the sentence requires a universally quantified noun phrase in subject position (15).

15. Every student is enrolled in at most 4 units.

Alternatively, a conditional sentence like (16) consisting of a dependent clause (expressing the condition) and a main clause (expressing the consequent) can be used to make the same statement.

16. If there is a student then the student is enrolled in at most 4 units.

Combining weak and strong negation in the same conditional sentence (17) allows us to express the closed-world assumption with respect to a given atom; meaning that all students who are not enrolled in a unit are explicitly known after processing the corresponding ASP rule.

17. If a student is not provably enrolled in a unit then the student is not enrolled in that unit.
18. If a person is holding an object at a time point and the person delivers that object at the same time point then the person will no longer be holding the object afterwards.

The conditional sentence (18) is interesting, since it describes an effect axiom (Mueller, 2015) for a temporal PENG^{ASP} specification. The condition contains a verb in present continuous tense that denotes a state and a verb in present tense that denotes an event. The consequent uses a verb in future continuous tense that describes the effect of the axiom.

5.3 Enforcing Constraints

In PENG^{ASP} constraints can be used to enforce conditions in a specification that must not become true. Syntactically, a constraint like (19) starts with a keyphrase (in brackets below for illustration purposes), followed by a potentially composite sentence.

19. [It is not the case that] a student who is enrolled in COMP3160 arrives at 11:00.

This composite sentence can have the same syntactic form as a sentence that can occur in the condition of a conditional sentence. In our case, the keyphrase is followed by a complex sentence that contains an embedded relative clause.

5.4 Asking Questions

PENG^{ASP} distinguishes between yes-no questions and wh-questions. Yes-no questions are formed in the same way as simple sentences, except that they have one auxiliary verb that occurs before, rather than after, the subject noun phrase, for example (20) and (21). Switching the placement of the auxiliary verb and the subject is called subject-aux inversion.

20. Is John enrolled in COMP3160?
21. Do most students work?

The formation of wh-questions involves interrogative words (*wh*-words and *how*). We distinguish in PENG^{ASP} between subject questions and complement/adjunct questions. Subject questions such as (22) and (23) are constructed from a wh-word and a finite verb phrase. Complement/adjunct questions such as (24) and (25) are formed from a wh/how-word that has been moved to the front and acts now as a filler for a gap in a subject-aux-inverted clause.

22. Who is enrolled in COMP3220?
23. Who is not enrolled in COMP3220?
24. What does Liam study?
25. When does the student arrive?

Note that we can answer question (23) in the context of our motivating example in Section 4, after adding the conditional sentence (17) to that specification, since the corresponding ASP rule for (17) ensures that all negative atoms for the enrollment property are derived. Answering the yes-no question in (21) requires a similar mechanism with an agreed threshold for the quantifier *most*.

A special case are questions that ask for an amount like (26) or a quantity like (27). They are formed with the help of a keyphrase and a noun that serve as a filler for a complement gap in a subject-aux inverted clause.

26. [How much] time does Liam spend on the first assignment?
27. [How many] units does Liam attend?

In the case of (26) the keyphrase is followed by an uncountable noun and in the case of (27) by a countable plural noun.

5.5 Issuing Directives

Directives are used in PENG^{ASP} to issue weak constraints in order to prioritise certain solutions. Syntactically, a directive is expressed with the help of a keyphrase like in (28) or (29) that starts with a specific verb in its bar infinitive form, followed by a priority level expressed as a prepositional phrase and a relative pronoun.

28. [Minimise with a priority of 3 that] a student accommodation is noisy.
29. [Maximise with a priority of 2 that] a student accommodation is central.

The description of the statement that is prioritised can have the same syntactic form as the description of a statement in a strong constraint.

6 Implementation Details

The grammar rules for the PENG^{ASP} system are specified in definite clause grammar notation and contain feature structures as arguments. These feature structures have the form of *name:value* pairs; names are Prolog atoms and values are Prolog terms (even compound terms of the form `[H|T]-T` are allowed).

The most important feature names in the grammar are: `mode` for the processing mode; `clause` with an incoming and an outgoing list as values for the assembly of ASP clauses (in the case of processing) and the disassembly of ASP clauses (in the case of generation); `ante` with an incoming and an outgoing list as values for the recording of accessible antecedents; `ctx` with a value (e.g., `fact` indicating a factual statement) for the functional context of a rule; and `fcn` with a value (e.g., `tmod` indicating a temporal modifier) for the structural function of a rule.

The feature structures for the functional context and the structural function of rules allow us to tailor the grammar for application scenarios that do not require the full power of ASP. This means the grammar is highly configurable and we can easily exclude certain linguistic constructions from the grammar if they are not required.

A number of additional syntactic feature names (e.g., `crd`, `num`, `vmode`, and `wform`) and semantic feature names (e.g., `def`, `arg` and `lit`) are used in the grammar; the meaning of these features should become clear in the following discussion.

The implementation of the bi-directional grammar can be best explained with the help of a concrete example. The grammar rules in Listing 4-10 translate a factual statement with a complex temporal modifier like:

30. Rona arrives on 2021-04-24 at 09:15.

incrementally into the following internal ASP representation:

Listing 2: Internal Answer Set Program

```
[['.',
  data_prop(A, 9, 15, time),
  data_prop(A, 2021, 4, 24, date),
  data_prop(A, B, date_time),
  happens(event(C, arrive), B),
  named(C, rona)]]
```

This internal ASP representation is then further translated by the Writer module of the PENG^{ASP} system into an executable ASP program:

Listing 3: Executable Answer Set Program

```
named(1, rona).
happens(event(1, arrive), 1619255700).
data_prop(2, 1619255700, date_time).
```

In the case of verbalising the above-mentioned ASP program, the Reader module reads this executable ASP program and expands it into the internal ASP representation but now in reverse order compared to the representation in Listing 2. The Planner module supports this process, if required, and decides when two or more literals should be aggregated and how these literals should be transformed into the aggregated structure.

6.1 Processing a Specification

For the processing of a specification, the `s`-rule in Listing 4 is used to split the sentence (30) into a noun phrase and a verb phrase. The feature structure `mode:M` takes care of the mode (either `proc` for processing or `gen` for generating). The feature structure `ctx:fact` indicates that this grammar rule is used to deal with a factual statement. The feature structure `clause:C1-C4` consists of a variable `C1` for the incoming list and a variable `C4` for the outgoing list. Remember that this data structure is used to collect the literals for the ASP program during the parsing process. This means the noun phrase takes a list as input and returns a modified list as output as indicated by the feature structure `clause:C1-C2`. This modified list then serves as input to the verb phrase as indicated by the feature structure `clause:C2-C3` and its output as input to the category for the full stop as indicated by the feature structure `clause:C3-C4`, since the full stop is the last element that is added to the front of the outgoing list (as illustrated in Listing 2). In a similar way, the feature structure `ante:A1-A3` collects all accessible antecedents with the help of an incoming and an outgoing list. The feature with the name `tree` takes a list as value and is responsible for constructing a syntax tree; this is in particular helpful for developing the grammar. It is important to note that all the above-mentioned tasks occur in parallel due to the power of unification.

Let us have a closer look at the noun phrase in the `s`-rule: the feature structure `crd:'-'` specifies that this noun phrase cannot be coordinated; the feature structure `fcn:subj` states that the noun phrase occurs in subject position and the feature structure `def:_` indicates that the definiteness of the noun phrase is unspecified in our example.

Listing 4: DCG rule for a factual statement

```
s([mode:M, ctx:fact, clause:C1-C4, ante:A1-A3, tree:[s:17, NP, VP]]) -->
  np([mode:M, ctx:fact, crd:'-', fcn:subj, def:_D, num:N, arg:X, clause:C1-C2,
      ante:A1-A2, tree:NP]),
  vp([mode:M, ctx:fact, crd:'+', num:N, arg:X, clause:C2-C3, ante:A2-A3, tree:VP]),
  fs([mode:M, clause:C3-C4]).
```

Listing 5: DCG rule for a noun phrase in subject position

```
np([mode:M, ctx:fact, crd:'-', fcn:subj, def:'+', num:N, arg:X, clause:C1-C3,
    ante:A1-A3, tree:[np:72, PN]]) -->
  pn([mode:M, wform:_, num:N, arg:X, clause:C1-C2, ante:A1-A2, tree:PN]),
  { anaphora_resolution(pn, [M, '+', X, C1, C2, C3, A1, A2, A3]) }.
```

Listing 6: DCG rules for the lexicon look up of a proper name

```
pn([mode:proc, wform:WForm, num:N, arg:X, clause:[C1|C2]-[[L|C1]|C2],
    ante:[A1|A2]-[[L|A1]|A2], tree:[pn:323, WForm]]) -->
  { lexicon([cat:pn, wform:WForm, num:N, arg:X, lit:L]) }, WForm.

pn([mode:gen, wform:WForm, num:N, arg:X, clause:[[L|C1]|C2]-[C1|C2],
    ante:[A1|A2]-[[L|A1]|A2], tree:[pn:324, WForm]]) -->
  { lexicon([cat:pn, wform:WForm, num:N, arg:X, lit:L]) }, WForm.
```

Listing 7: DCG rule for a verb phrase with a prepositional (temporal) modifier

```
vp([mode:M, ctx:fact, crd:'-', num:N, arg:X, clause:C1-C3,
    ante:A1-A3, tree:[vp:211, VP, PP]]) -->
 vc([mode:M, ctx:fact, num:N, arg:X, hold:[pred(X, PN)]-[happens(event(X, PN), T)],
     clause:C1-C2, ante:A1-A2, tree:VP]),
 pp([mode:M, ctx:fact, crd:'-', fcn:tmod, arg:T, clause:C2-C3, ante:A2-A3, tree:PP]).
```

Listing 8: DCG rule for an intransitive verb

```
vc([mode:M, ctx:fact, num:N, arg:X, hold:L1-L2, clause:C,
    ante:A-A, tree:[vc:217, IV]])  -->
  iv([mode:M, wform:_, num:N, vform:fin, arg:X, hold:L1-L2, clause:C, tree:IV]).
```

Listing 9: DCG rules for the lexicon lookup of an intransitive verb

```
iv([mode:proc, wform:WForm, num:N, vform:V, arg:X, hold:[L1]-[L2],
    clause:[C1|C2]-[[L2|C1]|C2], tree:[iv:325, WForm]]) -->
  { lexicon([cat:iv, wform:WForm, num:N, vform:V, arg:X, lit:L1]) }, WForm.

iv([mode:gen, wform:WForm, num:N, vform:V, arg:X, hold:[L1]-[L2],
    clause:[[L2|C1]|C2]-[C1|C2], tree:[iv:326, WForm]]) -->
  { lexicon([cat:iv, wform:WForm, num:N, vform:V, arg:X, lit:L1]) }, WForm.
```

Listing 10: DCG rule for a prepositional (temporal) modifier

```
pp([mode:M, ctx:fact, crd:'-', fcn:tmod, arg:T, clause:C1-C3,
    ante:A1-A3, tree:[pp:243, Prep1, Date, Prep2, Time]]) -->
 prep([mode:M, wform:[on], tree:Prep1]),
 date([mode:M, ctx:fact, arg:X, arg:T, clause:C1-C2, ante:A1-A2, tree:Date]),
 prep([mode:M, wform:[at], tree:Prep2]),
 time([mode:M, ctx:fact, arg:X, clause:C2-C3, ante:A2-A3, tree:Time]).
```

The feature structure `num:N` enforces number agreement between the noun phrase and the verb phrase, and the feature structure `arg:X` ensures that the argument for the noun (or proper name) in the noun phrase becomes available for the predicate-argument structure in the verb phrase. In contrast to the noun phrase that cannot be coordinated, the feature structure `crd:'+'` in the verb phrase indicates that verb phrase coordination is possible.

The `np`-rule in Listing 5 first checks whether the noun phrase that occurs in subject position consists of a proper name. For this purpose, the first `pn`-rule in Listing 6 is used that looks up the word form for the proper name in the lexicon and unifies the variable `L` for the literal with the corresponding value `named(C, rona)` that is stored in the lexicon for the proper name. The value for the literal is then added to the outgoing list that is responsible for clause construction (`clause`); at the same time the outgoing list that records all the accessible antecedents (`ante`) is updated. Afterwards, the anaphora resolution algorithm in the `np`-rule in Listing 5 is used to check if the proper name has been previously introduced and is now used anaphorically or not (both the clause lists and antecedent lists are then updated accordingly).

Once this has been done, the `vp`-rule in Listing 7 takes care of the verb phrase. This rule basically splits a verb phrase into an obligatory part and an optional part. The obligatory part consists in our case of an intransitive verb (without a complement) and the optional part consists of a prepositional phrase that serves as a temporal modifier for the verb. The important point to note here is that the `vc`-rule transforms a literal for an atemporal specification `pred(X, PN)` into a literal for a temporal specification `happens(event(X, PN), T)` with the help of a holding list, once the temporal modifier has been processed. The `vc`-rule in Listing 8 calls the first `iv`-rule in Listing 9 that processes the intransitive verb. Note that the variable `L1` that represents the literal for the intransitive verb is not immediately added to the outgoing list for the clause in this rule. This variable is first added to a holding list together with a second variable `L2` that serves as a placeholder. This second variable is added to the outgoing clause list instead of the first one. This is done because we do not known at this point of processing if the verb will finally be temporally modified or not. This information becomes only available once the `pp`-rule for the temporal modifier in Listing 10 has been processed. This rule adds three literals to the outgoing clause list (see Listing 2 for details) and completes the processing of the verb phrase.

6.2 Verbalising an ASP Program

Before an ASP program can be verbalised, it needs to be transformed into a linguistically processable version by the Reader module and potentially redundant structures need to be identified and aggregated by the Planner module. The grammar then takes the representation of the ASP program in Listing 2 in reverse order as input. The second `pn`-rule in Listing 6 removes the literal `named(C, rona)` from the incoming clause list and adds this literal to the outgoing antecedent list, since it may serve as a potential antecedent later. The same rule then generates the word form *Rona* for the removed literal. The anaphora resolution algorithm of the `np`-rule in Listing 5, then checks the status of the antecedent. Once the noun phrase has been generated, the second `iv`-rule in Listing 9 removes the (reduced) literal for the intransitive verb from the incoming clause list with the help of the information on the holding list and generates the verb form *arrives*. In a similar way, the three incoming literals for the temporal modifier are removed from the incoming clause list and generate the word forms that describe the temporal modifier. Generating a full stop terminates this process.

7 Conclusion

PENG^{ASP} is a machine-oriented controlled language designed to specify ASP programs in a natural way. The grammar of PENG^{ASP} is written in definite clause grammar notation and is a bi-directional one. The grammar can be used to translate a specification into an executable ASP program and to generate a semantically equivalent verbalisation of that ASP program. Anaphoric references can be resolved directly during the parsing process, since the anaphora resolution algorithm is tightly integrate with the grammar. The grammar is parameterised using feature structures so that subsets of the grammar can be selected in an easy way for various application scenarios without breaking the grammar. While this paper focuses on the features and coverage of the language and the grammar of PENG^{ASP}; it is important to note that the writing of a specification in PENG^{ASP} is supported by a smart authoring tool.

References

Peter Clark, Phil Harrison, Tom Jenkins, John Thompson, and Rick Wojcik. 2005. Acquiring and using world knowledge using a restricted subset of english. In *Proceedings of FLAIRS'05*, pages 506–511.

Esra Erdem, Michael Gelfond, and Nicola Leone. 2016. Applications of answer set programming. *AI Magazine*, 37(3):53–68.

Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. 2008. Attempto Controlled English for knowledge representation. In *Reasoning Web*, volume 5224 of *LNCS*, pages 104–124. Springer.

Gerald Gazdar and Chris Mellish. 1989. *Natural Language Processing in PROLOG, An Introduction to Computational Linguistics*. Addison Wesley.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, Sven Thiele, and Philipp Wanko. 2019. *Potassco User Guide, Version 2.2.0*.

Michael Gelfond and Yulia Kahl. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents, The Answer-Set Programming Approach*. Cambridge University Press.

Michael Gelfond and Valdimir Lifschitz. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming (ICLP)*, pages 1070–1080.

Stephen Guy and Rolf Schwitter. 2017. The PENG^{ASP} system: Architecture, language and authoring tool. *Journal of Language Resources and Evaluation, Controlled Natural Language*, 51:67–92.

Richard I. Kittredge. 2003. Sublanguages and controlled languages. In Ruslan Mitkov, editor, *The Oxford Handbook of Computational Linguistics.*, chapter 23, pages 430–447. Oxford University Press, Oxford.

Robert Kowalski. 2020. Logical english. In *Proceedings of the 2nd Workshop on Logic and Practice of Programming (LPOP)*, pages 33–37.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.

Erik T. Mueller. 2015. *Commonsense Reasoning: An Event Calculus Based Approach*. Morgan Kaufmann; Second Edition.

Fernando C.N. Pereira and David H.D. Warren. 1980. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278.

Rolf Schwitter. 2010. Controlled natural language for knowledge representation. In *Proceedings of COLING 2010*, pages 1113–1121. Association for Computational Linguistics.

Rolf Schwitter. 2020. Lossless semantic round-tripping in PENG^{ASP}. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5291–5293. International Joint Conferences on Artificial Intelligence Organization. Demos.

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. 2012. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96.

RegelSpraak: a CNL for Executable Tax Rules Specification

Mischa Corsius
HAN University of Applied Science
P.O. Box 2217, 6802 CE Arnhem
The Netherlands
mischa.corsius@han.nl

Stijn Hoppenbrouwers
HAN University of Applied Science
P.O. Box 2217, 6802 CE Arnhem
The Netherlands
Radboud University Nijmegen
P.O. Box 9010, 6500 GL, Nijmegen
The Netherlands
stijn.hoppenbrouwers@han.nl

Mariette Lokin
Ministry of Finance
P.O. Box 20201, 2500 EE
The Hague
The Netherlands
m.h.a.f.lokin@minfin.nl

Elian Baars
Dutch Tax Administration
John F. Kennedylaan 8, 7314 PS, Apeldoorn
The Netherlands
ejh.baars@belastingdienst.nl

Gertrude Sangers-Van Cappellen
Dutch Tax Administration
John F. Kennedylaan 8, 7314 PS, Apeldoorn
The Netherlands
gb.sangers-van.cappellen@belastingdienst.nl

Ilona Wilmont
HAN University of Applied Science
P.O. Box 2217, 6802 CE Arnhem
The Netherlands
Radboud University Nijmegen
P.O. Box 9010, 6500 GL, Nijmegen
The Netherlands
ilona.wilmont@han.nl

Abstract

RegelSpraak is a CNL developed at the Dutch Tax Administration (DTA) over the last decade. Keeping up with frequently changing tax rules poses a formidable challenge to the DTA IT department. RegelSpraak is a central asset in ongoing efforts of the DTA to attune their tax IT systems to automatic execution of tax law. RegelSpraak now is part of the operational process of rule specification and execution. In this practice-oriented paper, we present the history of RegelSpraak, its properties and the context of its use, emphasizing its double functionality as a language readable by non-technical tax experts but also directly interpretable in a software generating setup.

1 Introduction

The Dutch Tax Administration (DTA) is responsible for levying and collecting taxes in the Netherlands. This task comprises a diverse range of taxes, concerning both civil and business tax payers. Changes in tax legislation are carried out yearly, following the annual budgeting process. Every change has to be implemented in the relevant IT-applications, updates of instructions, and communications for tax lawyers within the DTA and for tax payers. This calls for a precise and unambiguous representation of the meaning of tax legislation.

Because the DTA's IT-landscape evolved over the years, various developing methods and tools and various programming languages are still used. On the route from law to automated execution, many interpretations and translations of legislation are produced. These translations are created by a variety of units and teams and are used in various ways. Consequently, there is a considerable risk of faults and of fragmentation of knowledge representations.

The teams of tax and IT-specialists involved in the implementation of legislation noted these issues and expressed the wish for a more coherent approach (Ausems, 2009). This wish was supported by the management, in view of their business goals: time savings and reduction of the risk of mistakes through reuse of rules. In an interview with a DTA stakeholder the common goal was formulated as follows: "*We need an approach that allows us to create a reusable specification of rules (Single Point of Definition), based on the text of legislation and policy rules and validated by tax experts, enabling an automatic translation to high power execution code*."

For example, in the future, the online calculations for income tax on the DTA-website and the web-portals for submitting yearly tax returns will use the same reusable components. Satisfaction of this requirement started with introduction of a new way of working, and a specific rule language: ***RegelSpraak***.

In this paper, we describe the history, the context of use, the users, and the main structure of RegelSpraak, finishing with recommendations for further development and improvement. We believe that RegelSpraak, being a fully operational CNL, is an interesting and rather unique artefact, given its double functional load: it is both human-understandable and indirectly, but fully, executable. In the field and tradition of CNLs (Kuhn, 2009) (Schwitter and Fuchs, 1996), this balance has always been a challenge.[1] The DTA has tackled this challenge in practise and we like to demonstrate how this is done. At the end of this paper, we share the insights and lessons learned from this development process for future research and implementation. This paper is practice-oriented rather than academic in nature, nevertheless it can be a notable contribution to the CNL field and community.

2 History of RegelSpraak

In 2007 the data and rule experts from the DTA got together with the founders of RuleXpress (Rulearts, 2021), a dedicated tool for business rule specification and management. A small team started researching how this tool could be applied within the DTA. They started with Semantics of Business Vocabulary and Rules (SBVR) (Object Management Group, 2021) as a basis, but soon discovered that this standard was not fit for purpose because it did not provide a full set of expressions (i.e. it is not really a rule language, as such). They switched to RuleSpeak (Ross, 2006) as a viable alternative, as it works with informally constrained natural language text and is thus suitable for discussion between lawyers and IT-experts (Baars, 2009). RuleSpeak, however, does not impose a complete set of active syntactic constraints. Therefore, it allowed for too much freedom in rule specification, which limited its usefulness. The team started composing specific language patterns, still largely based on RuleSpeak structures, aiming for a more concise and completely constrained syntax. Consequently, a specific pattern schema was developed. The first version of the rule pattern document (Sangers - van Cappellen and Van Kleef, 2010) consisted of keywords for use in rules, and a number of patterns. These were based on RuleSpeak as much as possible.

The first patterns were written to convert existing rules for Tax Pre-Check (Dutch: Fiscale Voorcontrole) to a readable format. In those early days, MS Excel was used to provide a template for describing rules (see Figure 2). In order to properly and consistently convert the Tax Pre-Check rules to RegelSpraak rules, the DTA first went through the various Excel sheets to determine the different variants of the rules. Patterns and conventions were then drawn up for each of these rule variants. Next, all 2300+ rules had to be converted from Excel to RegelSpraak. A partly automated conversion was carried out. The code-like terms were translated into readable terms based on the Excel sheet and each sentence was converted to a rough RegelSpraak sentence. After that, a manual step was needed to make the rules fully compliant with the RegelSpraak format. The capture of the RegelSpraak (meta)rules was done in RuleXpress, the compilation and transformation in ANTLR. Parr (2014) offers a detailed description of the

[1] In Azzopardi et al. (2018) and also in Calafato et al. (2016), the use of CNL's in a tax and financial context in other countries is discussed.

ANTLR-parser. With this setup, the team started to build rules in XSD/XML.
For the construction and development of a grammar file, the ANTLR Works GUI workbench was used. A grammar and parser were iteratively developed for each of the domains. These parsers were used to check if the RegelSpraak specifications were syntactically correct. Once every possible RegelSpraak statement could be parsed, a compiler could be generated. The intention was to demonstrate that compilers could be made for various different programming languages (e.g. SRL, Java, C++). Consequently, this led to a different format of output for each compiler.
The next step in the development of RegelSpraak was the 'Proof of Concept *Health care insurance law'* (Dutch: Zorgverzekeringswet). In this PoC, for the first time, rules were written based on a direct analysis of the legislation. Thus, the natural language rules were not based on existing code rules but on the text of their legal sources. The analysis of the legislation was carried out along a number of questions that had to be answered for every part (article, section, subsection, sentence, formula) in the legislation. These questions were:

- For whom should something be determined? This is the [role] or [object] in the rule pattern.
- What must be determined? This concerns the [element_to_be_determined] in the rule pattern.
- When should this be determined? This is the [reference_date] in the rule pattern.
- Are there limit values that have to be taken into account? These are the [parameters] in the rule pattern.
- Which keywords are used in the text? These are the fixed natural language terms or phrases.

Several PoCs were carried out to test and refine RegelSpraak. In order to enable automated execution, extra constraints – in addition to Dutch grammar – were needed to achieve unambiguous rules. Parallel to these PoCs, the DTA invested in development of a method and tooling for structured analysis of legal texts, to achieve a solid and precise knowledge base underlying the RegelSpraak-rules. This resulted in iKnow Cognitatie (Cognitatie, 2021) Furthermore, tooling for validation and automatic transformation of the RegelSpraak-rules into code was developed, based on the MPS/Jetbrains workbench (Jetbrains, 2021), now known as the Agile Law Execution Factory (ALEF). Thus, a production chain for rule-based IT development was realized, as illustrated in Figure 7.

Today, RegelSpraak is broadly used in IT development at DTA and is subsequently expanded by newly added sections of the law, and by new patterns on the basis of experiences in applying the language. An important factor in the success of RegelSpraak is its deployment in multi-disciplinary teams which are an important driving force behind the ongoing development of RegelSpraak. The DTA has its own in-house training programme and documents new versions of language and tooling in detailed reference manuals. User evaluation was informally done in the past, see Wilmont et al. (2021) for a more formal method of evaluation.

3 Main Features of RegelSpraak

A basic rule pattern applies to all rules. A RegelSpraak rule always has the following format: [RESULT] IF [CONDITION(S)]. A condition compares attributes, which can have boolean or numerical values, or be dates, enumerations or roles. The result is executed as a consequence of the successful evaluation of the conditions. The results and conditions are connected using carefully composed Dutch phrases to maximize the resemblance to a natural sentence.

For legibility's sake, in our examples, (almost) all examples have been translated into English, without showing the original Dutch fragments.

```
IF (is filled in BALANCE_INCOME-COST_OWNER-OCCUPIED_HOME
[0182] OR
is filled in BALANCE_DECUCTIBLE_COST_OWNER-OCCUPIED_HOME
[0173])
THEN FILL BALANCE_INCOME-COST_OWNER-OCCUPIED_HOME [0172]
WITH
(BALANCE_INCOME-COST_OWNER-OCCUPIED_HOME [0182] -/-
BALANCE_DECUCTIBLE_COST_OWNER-OCCUPIED_HOME [0173])
```

Figure 1: Example Rule 1 without metadata

Rule number	Name	Repetition manner	Rule	Repetition	Consult? Use? Applying	Comment
DA 270	Balance_income_minus_cost _owner-occupied_home	Single	IF (is filled in BALANCE_INCOME-COST_OWNER-OCCUPIED_HOME [0182] OR	N	Single	Rule 1224: Balance income cost owner occupied home: This fact is determined in earlier deduction: the sequence is relevant! total deductible cost owner-occupied home: This fact is determined in earlier deduction: the sequence is relevant!
			is filled in BALANCE_DECUCTIBLE_COST_OWNER-OCCUPIED_HOME [0173])	N	Single	
			THEN FILL BALANCE_INCOME-COST_OWNER-OCCUPIED_HOME [0172] WITH	N	Single	
			(BALANCE_INCOME-COST_OWNER-OCCUPIED_HOME [0182] -/-	N	Single	
			BALANCE_DECUCTIBLE_COST_OWNER-OCCUPIED_HOME [0173])	N	Single	

Figure 2: Example Rule 1 with metadata (Excel-file)

The rule in the Excel file (see Figure 2) consists of two different parts, namely an IF and a THEN-part. RuleSpeak prescribes starting with the THEN-part, usually followed by an IF-part.
Rule 1 is a derivation rule, in which two elements are added up to create a third element. The two elements must not always be added up; the rule contains two conditions under which the calculation must be performed. In this case, one or both elements must be completed.

For an adding up of two elements, the wording MUST BE CALCULATED AS ... PLUS … was chosen, because this comes as close as possible to a natural language expression. The element that is derived is called [element_to_be_determined]. For the condition part, where one or both conditions must be true, IF ... SATISFIES AT LEAST ONE OF THE FOLLOWING CONDITIONS is used. Then the two conditions are mentioned, as an enumeration (the lines with indent in the example below). The rule pattern that fits the result part here is called "basic calculation". A basic calculation is one of the basic rule patterns in RegelSpraak.

As stated earlier, a basic rule pattern applies to all rules, i.e. most rules consist of two parts, namely a *result* part and a *condition* part.

The *result* part is the first part of the rule, describing how the variable that has to be determined gets its value. At the moment, specific rule patterns exist for the result part of derivation rules, restriction rules, classification rules, and process rules. The *condition* part is the part of the rule that describes the conditions under which the variable that has to be determined gets its value. The condition part of a rule starts with the word "if" (see Figure 4 and 6).

In RegelSpraak, rule 1 is expressed as follows:

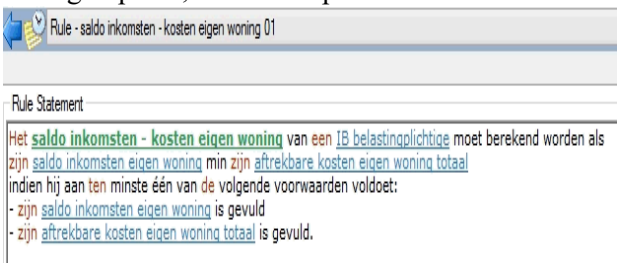


Figure 3: Example of RegelSpraak Rule 1 (Dutch)

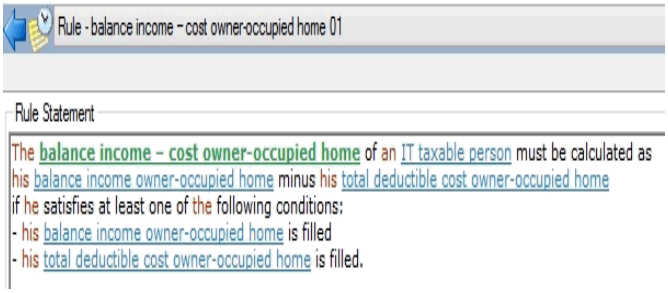


Figure 4: Example of RegelSpraak Rule 1 (English translation)

The basic rule patterns used in this rule are "basic calculation" and "disjunction". A basic calculation consists of a [value] PLUS/MINUS/MULTIPLY/DIVIDE BY [value]. In a basic calculation multiple operators and multiple values may occur. We will illustrate this in the two following examples:
[element_to_be_determined] MUST BE CALCULATED AS [value] PLUS/ MIN/ MULTIPLY/ DIVIDE BY [value]

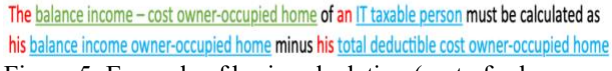


Figure 5: Example of basic calculation (part of rule 1)

… IF HE /A /THE /EACH PRESENCE OF ITS [object] SATISFIES AT LEAST

ONE/TWO/THREE/... OF THE FOLLOWING CONDITIONS

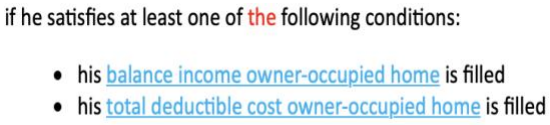


Figure 6: Example of disjunction (part of rule 1)

Rules are often similar in form but different in purpose. For example, there are rules expressing an addition of amounts, or formulas that select the highest amount of two. Rule patterns have been developed to ensure that rules with the same goal are composed in the same way. A rule pattern is a template for concrete rules. By filling this template with the correct elements, an unambiguous rule is created. Each rule must comply with one or more rule patterns.

4 Use and users

RegelSpraak now is a much appreciated and crucial tool in the DTA. The constructive and step-by-step research and development process has paid off, and gradual implementation in the organisation has created substantial commitment and adoption. This allowed the DTA to increase the number of employees working on RegelSpraak in various teams and roles, in a multidisciplinary setting.

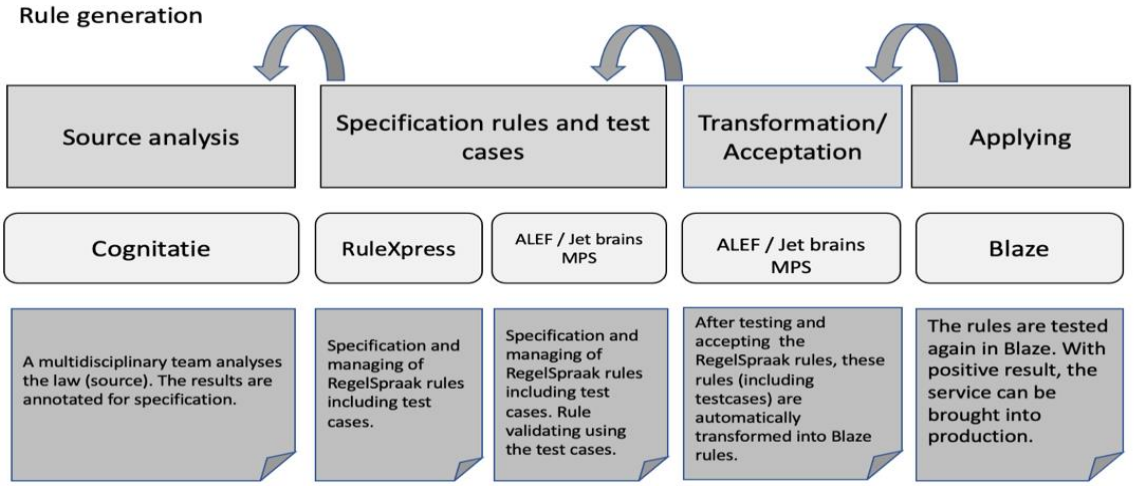


Figure 7: Rule generation

Figure 7 illustrates the rule generation process. It starts when a new or changed piece of legislation is adopted. Tax specialists, rule analysts and IT developers, working in a multidisciplinary team, analyse the piece of legislation in detail and draw up complementary test cases. They make annotations in the legislation using a legal analysing scheme, consisting of predefined labels for legal concepts in the legislation. The annotations are made in *iKnow Cognitatie* (Cognitatie, 2021), a tool especially developed for this. After that, the rule analysts specify rules based on the annotations. In some cases, RuleXpress is used for managing the rules and test cases. It provides repositories to do so. The next step in rule generating is importing this annotated law content into ALEF. In other cases, the rules are specified directly in ALEF. RuleXpress has a more user-friendly interface and better rule management functionality.

The test cases are also added to ALEF, so the rules can be tested. At this point, the tax expert comes in again and validates whether the rules yield the correct test outcomes in the test cases. Finally, the rules are automatically transformed into Blaze-code (Blaze being a high-performance proprietary rule engine (Blaze, 1999)), tested again with the test-cases, and deployed as a webservice.

This process allows for iteration; flaws are now often detected at an early stage in the process. An unambiguous representation, fully in accordance with the tax law, is better achievable. The output of this process is now primarily used as a

component in various back office IT systems, but it is also available for online tax declaration forms and calculation wizards on the website of the DTA.

5 Lessons learned and conclusion

What started off as a small exploration and gradually expanded to PoCs of CNLs in the DTA has now become a robust and scalable infrastructure which to our knowledge has no precedent. The DTA has successfully experimented with RegelSpraak and has gradually become leading within Dutch government in the development and use of CNL. Collaboration with other public organisations is in an exploratory stage, as acknowledgement is growing of the usefulness of shared, reusable rules as a basis for diverse IT applications, and as a central asset in the future of automated law execution.

In the past years, the DTA has documented the RegelSpraak patterns and conventions in manuals, reference guides and workflow-documents. A knowledge community had been created, as well as an in-house training programme. Nevertheless, there is still a lot of tacit knowledge in the heads of the employees, especially with respect to quality aspects of RegelSpraak patterns and rules. The current paper provides a limited overview of what was done so far to make this knowledge explicit. However, further work is needed to validate and improve the framework, patterns, and rule specification procedure. We intend to perform a user research of this, observing and analysing the work of various experts *with* and *on* RegelSpraak.

Lessons learned, for the DTA but also for other organisations implementing CNLs for law execution purposes, are the following:

- Start small: choose a well-scoped and limited part of legislation to start; this will provide an interesting showcase, creating crucial managerial backing.
- Put together a team which comprises the relevant expertise for your organisation.
- Find a sponsor at management level who shows commitment and provides you and your team with the necessary resources (time and tooling), and even more importantly: with space to fail every now and then.
- Invest in education, preferably by training on the job.
- Share knowledge early – pay attention to explicit communication and promotion by means of presentations and publications. This is a good way to share the methods and knowledge within and outside the organisation. The DTA gladly took opportunities to give (invited) presentations, for example at Business Rules Platform Nederland (BRPN) meetings.
- Start testing at once to quickly fix bugs and problems. This saves a considerable amount of money in the overall development project.

6 References

Ausems, A, et al. 2009. Nota Visie Metagegevens & Bedrijfsregels. Version 0.97. Unpublished.

Azzopardi, S, et al. 2018. A Controlled Natural Language for Financial Services Compliance Checking. In *Controlled Natural Language - Proceedings of the Sixth International Workshop*. Kildare, Ireland.

Baars, E. 2009. Nota Eindrapportage: Training Business Rules. Version 1.0. Unpublished.

Blaze software, 1999. Blaze advisory, California. [accessed 2021 May 12]. Retrieved from: https://condor.depaul.edu/jtullis/documents/Blaze_Advisor_Tech.pdf.

Calafato, et al. 2016. A Controlled Natural Language for Tax Fraud Detection. In *Controlled Natural Language – Proceedings of the 5th International Workshop*, Aberdeen, UK.

Cognitatie, the tool that makes your organization compliant and transparent. 2021. PNA Group. [accessed 2021 April 8]. https://www.pna-group.com/kennismanagement-software/iknow-cognitatie/.

Jetbrains. 2021. Jetbrains s.r.o; [accessed 2021 April 8]. https://www.jetbrains.com/mps/.

Kuhn, T. 2009. How to Evaluate Controlled Natural Languages. *In Workshop on Controlled Natural Language,* Marettimo, Italy.

Object Management Group. 2021. The Object Management Group® (OMG®); [accessed 2021 April 8]. https://www.omg.org/spec/SBVR/About-SBVR/.

Parr, T. 2014. ANTLR (Another Tool for Language Recognition). [accessed 2021 April 8]. https://www.antlr.org/.

Ross, R. 2006. The RuleSpeak® Business Rule Notation. In *Business Rules Journal Vol. 7, No. 4.* http://www.brcommunity.com/a2006/b282.html.

Rulearts: Business rules software has a name: RuleXpress. 2021. [accessed 2021 April 8]. http://www.rulearts.com/rulearts-products/rulexpress-business-rules-software/.

Sangers – van Cappellen, G. and Van Kleef, M. 2010. Modelleerpatronen en conventies Specificatieteam; version 0.3. Unpublished.

Schwitter, R. and Fuchs, N.E. (1996). Attempto: From Specifications in Controlled Natural Language towards Executable Specifications. In *EMISA Workshop 'Natürlichsprachlicher Entwurf von Informationssystemen - Grundlagen, Methoden, Werkzeuge, Anwendungen'*, Tutzing.

Wilmont, I., Dulfer, D., Hof, J., Corsius, M., Hoppenbrouwers, S. A quality evaluation framework for a CNL for agile law execution. In *CNL 2021: Seventh International Workshop on Controlled Natural Language*. Amsterdam, The Netherlands.

The Law of Inertia and the Frame Problem in Attempto Controlled English

Norbert E. Fuchs
Department of Computational Linguistics
University of Zurich
fuchs@ifi.uzh.ch
http://attempto.ifi.uzh.ch

Abstract

Daily experience teaches us that a situation remains unchanged unless somebody or something changes it. Leibniz called this experience the law of inertia. Early attempts to formalise the law of inertia failed because they offered no easy way to describe that after a partial change of a situation the unaffected rest remains unchanged. This so-called frame problem was efficiently solved by later approaches, specifically by the event calculus and the default logic. Focusing on default logic, I will show that it can express the law of inertia not only in first-order logic, but also quite naturally in Attempto Controlled English. Furthermore, I will use the Attempto reasoner RACE to efficiently reason with the law of inertia.

1 The Common Experience of Inertia

When you return to your office in the morning you expect to find the items on your desk in exactly the same order as you left them in the evening before – unless somebody or something moved them. This common experience was expressed by the philosopher Leibniz as the law of inertia: "Everything is presumed to remain in the state in which it is." (Leibniz, 1679). At about the same time Newton published his three laws of motion, the first of which expresses the specific case of the law of inertia for moving physical bodies (Newton, 1687).

I will show that Leibniz' law of inertia can be formalised in Attempto Controlled English (ACE)[1] and that this formalisation allows us to reason with the law. In section 2, I describe early attempts to formalise common sense, specifically the law of inertia, the encountered frame problem, and a variant of the Yale Shooting Problem. Section 3 presents two solutions of the frame problem, the event calculus and the default logic. In section 4, I express the default logic in ACE. Section 5 describes reasoning with the law of inertia using the Attempto reasoner RACE[2]. Section 6 shows that incorporating the law of inertia into RACE facilitates the reasoning. Section 7 revisits the Yale Shooting Problem in ACE/RACE. Section 8 summarises the paper and briefly addresses the fact that the law of inertia and, specifically, its formalisations are asymmetric with respect to time.

2 Formalising Common Sense and the Frame Problem

Beginning in the 1960s researchers began to formalise common sense, predominantly in first-order logic. When trying the express the law of inertia they encountered the so-called frame problem, that is how to effectively and efficiently describe the unaffected part of a situation after a partial change.

Initial attempts to solve the frame problem failed, mostly because only the changed parameters were taken into account and the unchanged parameters ignored.

The shortcomings were strikingly demonstrated by the impossibility to adequately solve the so-called "Yale Shooting Problem" (Hanks and McDermott, 1987). In this paper I replace the original problem by a less violent, but "problem-identical" one.

[1] http://attempto.ifi.uzh.ch/

[2] http://attempto.ifi.uzh.ch/race/

Initially, a wine glass is empty and a wine bottle is not open. Opening the bottle, waiting a moment to read the label, and then pouring the wine should fill the glass. If this situation is formalised in first-order logic by only taking the changed parameters into account and ignoring the unchanged ones, it cannot be uniquely proved that the wine glass is finally full. In one logical solution the wine glass is actually full; in another logical solution the wine bottle is again not open and the wine glass remains empty.

Here is a simple formalisation[3] of my version of the Yale Shooting Problem using four time points 0, 1, 2, 3 and the two fluents – conditions that can change their truth value over time – *empty* and *open* expressed by the following first-order formulas:

$$empty(0) \quad (1)$$

$$\neg\, open(0) \quad (2)$$

$$true \rightarrow open(1) \quad (3)$$

$$open(2) \rightarrow \neg\, empty(3) \quad (4)$$

As it turns out, two evaluations of the fluents are consistent with the formulas, the first one

{empty(0), ¬ open (0)}, {empty(1), open (1)}, {empty(2), open (2)}, {¬ empty(3), open (3)}

describing the intended behaviour that the glass is not empty at time 3, the second one

{empty(0), ¬ open (0)}, {empty(1), open (1)}, {empty(2),¬ open (2)}, {empty(3),¬ open (3)}

describing the non-intended behaviour that the bottle strangely is not open again at time 2 and the glass is empty at time 3.

The problem is that the formulas only describe the changes and that they do not specify that fluents unaffected by the actions remain unchanged. Several so-called frame axioms – e.g. *empty(1) = empty(0)* – would be needed to restrict the solutions to the one with the intended behaviour.

3 Solutions of the Frame Problem

In the end, several correct solutions for the frame problem and the Yale Shooting Problem were developed. Among the solutions that express Leibniz' law of inertia directly are the Event Calculus (Kowalski and Sergot, 1986) and the Default Logic (Reiter, 1980).

The Event Calculus uses the following conceptualisation: linear time, fluents F that hold or do not hold at time points, and events E that happen at time points and initiate or terminate fluents. There is one domain-independent axiom of inertia

holdsAt(F, T) ← happens(E1, T1) ∧ initiates(E1, F) ∧ T1< T ∧ ¬ ∃ E2, T2 [happens(E2, T2) ∧ terminates(E2, F) ∧ T1< T2 ∧ T2< T (5)

with the meaning "The fluent F holds at a time T if an event E1 happens at a time T1 before T and E1 initiates F and there is no event E2 and there is no time T2 between T1 and T so that E2 happens at T2 and E2 terminates F."

To describe a concrete situation, a set of domain-dependent axioms would be needed to specify the actual fluents and the actual events that initiate and terminate the fluents.

The Default Logic – which will be used in this paper – relies on non-monotonic logic with assumptions and exceptions expressed as default inference rules of the form

$$\frac{prerequisite : justification}{conclusion} \quad (6)$$

meaning "If the prerequisite is true and the justification is consistent with the known facts then the conclusion can be drawn."

As a concrete example here is the default inference rule for the law of inertia

$$\frac{r(X,S) : r(X,\ do(A,S))}{r(X,\ do(A,S))} \quad (7)$$

paraphrased as "If r(X) is true in a situation S and it can be assumed that r(X) remains true after the action A is applied to S then r(X) remains true."

Being inspired by (Erdem et al., 2016), I reformulated (Equation 7) as an implication using logical negation (¬) and negation as failure (*not*).

r(X,S) ∧ not (¬ r(X, do(A,S))) → r(X, do(A,S)) (8)

with the paraphrase "If r(X) is true in a situation S and it is not provable that r(X) is false after the action A is applied to S then r(X) remains true."

[3] adapted from https://en.wikipedia.org/wiki/Yale_shooting_problem

Replacing the situation parameter by time points we get

$r(X,T1) \wedge T2>T1 \wedge not\ (\neg\ r(X,T2)) \rightarrow r(X, T2))$ (9)

that is "If r(X) is true at a time T1 and there is a later time T2 and it is not provable that r(X) is false at T2 then r(X) is true at T2."

4 Default Logic in Attempto Controlled English

Default logic's basic axiom for inertia (Equation 9) can be paraphrased in Attempto Controlled English (ACE) as

If something X holds at a time T1 and there is a time T2 that is after T1 and it is not provable that X does not hold at T2 then X holds at T2. (10)

Note that I introduced a practically identical axiom when briefly discussing the frame problem in (Fuchs, 2016).

As a concrete example let's model the situation of a sleeping person that can be expressed in English as "If a person falls asleep and the person does not wake then the person continues to sleep.", ignoring the duration and nature of human sleep to solely focus on the law of inertia. Using verbs of the sleep example (*fall asleep*, *wake*, *sleep*) directly instead of the verb *hold* we can customise (Equation 10) for the sleep example

If a person falls asleep at a time T1 and a time T2 is after T1 and it is not provable that the person wakes at T2 then the person sleeps at T2. (11)

or simply using only the verb *sleep*

If a person sleeps at a time T1 and a time T2 is after T1 and it is not provable that the person does not sleep at T2 then the person sleeps at T2. (12)

5 Reasoning with in Attempto Controlled English

Having default logic's basic axiom for inertia expressed in ACE we may want to reason with it. There are several reasoners for ACE of which I will use the Attempto reasoner RACE (Fuchs, 2012; Fuchs, 2016). RACE can

- determine the (in-) consistency of an ACE text
- deduce one ACE text from another one
- answer an ACE query from an ACE text

RACE is supported by about 100 auxiliary axioms – expressed in Prolog, using ACE's internal logical notation – that provide domain-independent knowledge like the relations between plural and singular nouns, the relations between numbers, the substitutions for query words, and much else. In spite of their large number auxiliary axioms have little impact on RACE's performance since they are only called individually and only when needed.

RACE is implemented in Prolog, has a web-service and a web-interface whose output window will be used in the following.

RACE has a restriction relevant for the problem at hand: For technical reasons RACE does not accept logical negation within the scope of negation as failure (*it is not provable that ... not ...*). Thus, we will have to replace negated verbs within negation as failure by verbs that express the intended negation, e.g. *does not sleep* → *wakes*.

Reasoning Examples: Continuous Sleep and Interrupted Sleep. In the following RACE will prove that a person whose sleep is not interrupted will continue to sleep, while a person whose sleep is interrupted does no longer sleep.

First the case of continuous sleep. To the axiom of the law of inertia of a sleeping person (Equation 11) two axioms are added to describe a concrete situation: one axiom to introduce a person sleeping at an initial time and a second axiom to introduce a later time. The theorem checks whether the person will sleep at the later time.

If a person sleeps at a time T1 and a time T2 is after T1 and it is not provable that the person wakes at the time T2 then the person sleeps at the time T2.

A person sleeps at an initial time. A later time is after the initial time.
⊢
A person sleeps at a later time.

Submitting this reasoning example to RACE's web-interface we get the expected result (Figure 1) that RACE proves that the person sleeps at a later time. To present the result, I use a screenshot of the output window of RACE's web-interface. This window contains the axioms, the theorem, and the minimal subset of the axioms needed to prove the theorem. The entry "parameters" is used for testing.

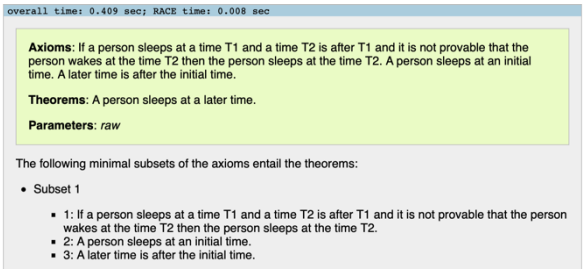


Figure 1: Continuous Sleep

Following is the case of interrupted sleep. To the axiom of the law of inertia of a sleeping person (Equation 11) four axioms are added: one axiom relates waking to not sleeping, one axiom introduces a person sleeping at an initial time, one axiom introduces a later time, and one axiom states that the person wakes at the later time. The theorem checks whether the person will sleep at the later time.

If a person sleeps at a time T1 and a time T2 is after T1 and it is not provable that the person wakes at the time T2 then the person sleeps at the time T2.

If a person wakes at a time T then the person does not sleep at the time T.

A person sleeps at an initial time. A later time is after the initial time. The person wakes at the later time.

⊢

A person sleeps at a later time.

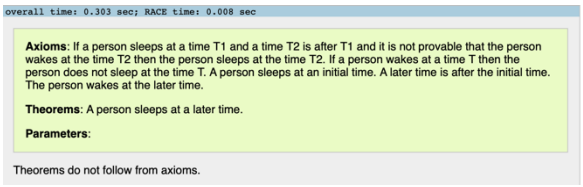


Figure 2: Interrupted Sleep

Submitting this reasoning example to RACE we get the expected result (Figure 2) that RACE cannot prove that the person sleeps at the later time.

6 Incorporating the Law of Inertia into RACE

For a complex situation involving many fluents we would have to formulate for each fluent a separate ACE axiom of inertia thus blowing up the axiomatisation of the situation. Going back to ACE's general axiom of inertia (Equation 10) offers no solution since for each fluent of the concrete situation we would have to introduce bridging axioms – as in the event calculus – thus again creating a blow-up. Instead, I decided to incorporate an abstract version of (Equation 10) as three auxiliary Prolog axioms into RACE: one axiom for intransitive verbs, one axiom for transitive verbs, and one axiom for the copula plus adjective. This threefold division is necessary since these verbs have different internal representations. There is no axiom for ditransitive verbs because they do not seem to cause the frame problem. By abstracting away from the details of the fluents – concretely replacing nouns, verbs etc. by variables – these three auxiliary axioms can cover any fluent. One could say that RACE now "knows" the law of inertia in the same way as it "knows" the relation between plural and singular nouns.

There is an unexpected bonus: the new auxiliary axioms can deal with logical negation within the scope of negation as failure (*it is not provable that ... not ...*), eliminating the need to replace negated verbs by other verbs.

Let us now return to the previous examples.

Reasoning Examples: Continuous Sleep and Interrupted Sleep Revisited. In the following RACE will again prove that a person whose sleep is not interrupted will continue to sleep, while a person whose sleep is interrupted does no longer sleep.

Note that the verb *wake* of the initial example can now be replaced by *not sleep*.

Further note that in both cases the ACE axiom expressing inertia (Equation 12), is no longer necessary for the proof. However, it is left crossed out as a reminder to the reader.

Following is the case of the continuous sleep.

~~*If a person sleeps at a time T1 and a time T2 is after T1 and it is not provable that the person does not sleep at the time T2 then the person sleeps at the time T2.*~~

A person sleeps at an initial time. A later time is after the initial time.

⊢

A person sleeps at a later time.

We get the expected result (Figure 3) that – using the (hidden) auxiliary axiom "Frame Axiom 1: Persistence of intransitive verb." in addition to the (visible) axioms describing the concrete situation – RACE proves that the person sleeps at a later time.

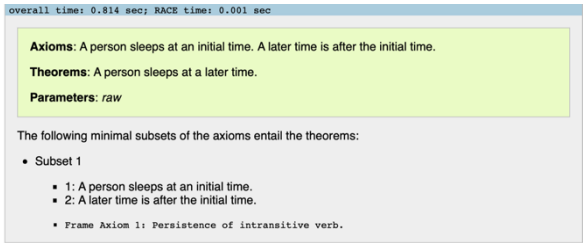


Figure 3: Continuous Sleep Revisited

Now the case of interrupted sleep. Note that also the axiom relating waking to not sleeping, that I used previously, is no longer needed.

~~*If a person sleeps at a time T1 and a time T2 is after T1 and it is not provable that the person does not sleep at the time T2 then the person sleeps at the time T2.*~~

A person sleeps at an initial time. A later time is after the initial time. The person does not sleep at the later time.

⊢

A person sleeps at a later time.

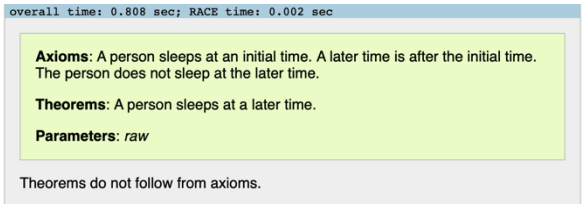


Figure 4: Interrupted Sleep Revisited

We get the not at all surprising result (Figure 4) that RACE cannot prove that the person sleeps at a later time.

7 The Yale Shooting Problem Revisited

RACE can now correctly and efficiently solve my version of the Yale Shooting Problem. Here again the version in English

A wine glass is initially empty and a wine bottle is initially not open. Opening the bottle, waiting a moment and then pouring the wine should fill the glass.

and here the ACE version

A glass is empty at a time T0 and a bottle is not open at T0. A time T1 is after T0 and the bottle is open at T1. An intermediate time T2 is after T1. A final time T3 is after T2. If a glass is empty at T2 and a bottle is open at T2 then the glass is not empty at the final time T3.

⊢

There is a final time. A glass is not empty at the final time.

In addition to the three frame axioms introduced before, a fourth frame axiom is needed to relate the state of some fluents to the precondition of an implicative axiom, concretely the fluents "*empty glass*" and "*open bottle*" at the time T2 to the precondition of the implicative axiom "*If a glass is empty …*".

Submitted to RACE we get:

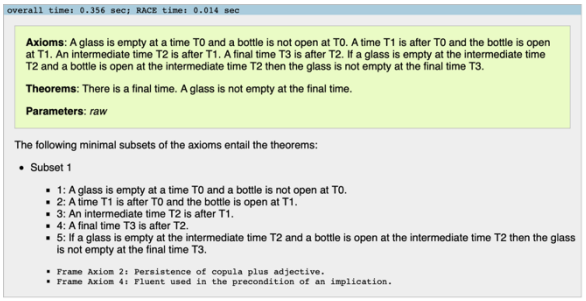


Figure 5: Yale Shooting Problem Revisited

The Yale Shooting Problem presented in section 2 had two solutions, one expected, the other one unexpected. RACE – using its built-in inertia axioms "Frame Axiom 2: Persistence of copula plus adjective." and "Frame Axiom 4: Fluent used in the precondition of an implication." – generates only the expected one.

8 Conclusions

I arrive at the following conclusions

- default logic effectively formalises the omnipresent law of inertia,
- default logic – originally expressed in first-order logic – can be naturally formulated in Attempto Controlled English (ACE),
- the Attempto reasoner RACE can reason with the law of inertia,
- reasoning with the law of inertia can be simplified and generalised by expressing the basic axioms not in ACE but as auxiliary Prolog axioms, quasi incorporating the law of inertia into RACE,
- RACE can correctly solve the Yale Shooting Problem.

Leibniz' law of inertia connects the present implicitly to the future, it does not make any assump-

tions about a possible past. This is explicitly reflected in the formalisations presented. To reason backwards in time, one would need other approaches, for instance abduction, a simple form of which is found in (Fuchs, 2016).

Acknowledgments

I enjoyed intensive discussions with Bob Kowalski and Rolf Schwitter which motivated me to investigate the formalisation of the law of inertia in Attempto Controlled English. Many thanks go to the first reviewer who not only carefully put this paper into the context of my previous publications, but also made valuable suggestions for clarifications, extensions and future research. I would also like to thank the Department of Computational Linguistics, University of Zurich, for its hospitality.

References

E. Erdem, M. Gelfond and N. Leone. 2016. *Applications of Answer Set Programming*. ACM. (www.depts.ttu.edu/cs/research/documents/5.pdf)

N. E. Fuchs. 2012. *First-Order Reasoning for Attempto Controlled English.* In *Proc. of the Second International Workshop on Controlled Natural Language (CNL 2010).* Marettimo, Italy.

N. E. Fuchs. 2016. *Reasoning in Attempto Controlled English: Non-monotonicity.* In *Proc. of the Fifth International Workshop on Controlled Natural Language (CNL 2016).* Aberdeen, UK.

S. Hanks and D. McDermott. 1987. *Nonmonotonic logic and temporal projection*. Artificial Intelligence, 33(3):379–412.

R. Kowalski and M. Sergot. 1986. *A logic-based calculus of events*. New Generation Computing, 4 (1): 67–95 (1986).

G. Leibniz. 1679. *An Introduction to a Secret* Encyclopaedia.

I. Newton. 1687. *Philosophiæ Naturalis Principia Mathematica*.

R. Reiter. 1980. *A logic for default reasoning*. Artificial Intelligence, 13:81-132 (1980).

Reasoning in Attempto Controlled English: Mathematical and Functional Extensions

Norbert E. Fuchs
Department of Computational Linguistics
University of Zurich
fuchs@ifi.uzh.ch
http://attempto.ifi.uzh.ch

Abstract

RACE is a first-order reasoner for Attempto Controlled English (ACE). This paper introduces mathematical and functional extensions. It is the third system description of RACE, and also the final one since RACE now covers all ACE constructs that have a representation in first-order logic.

1 Introduction

Attempto Controlled English (ACE)[1] is a logic-based knowledge representation language that uses the syntax of a subset of English. The Attempto Reasoner RACE[2] allows users to show the consistency of an ACE text, to deduce one ACE text from another one, and to answer ACE queries from an ACE text.

Two previous system descriptions (Fuchs, 2012; Fuchs, 2016) of RACE detailed its structure, its functionality, its implementation and its user interfaces, material that will repeated here only to the extent to make this paper self-contained.

This is the third – and final – system description of RACE intended to complete its coverage of ACE. Concretely, RACE has been extended to reason with ACE's mathematical and functional constructs. The mathematical extension offers primarily solutions for arithmetic problems and linear equations. The functional extension allows ACE to directly access Prolog predicates which has a number of important consequences, for example the ability to express recursive algorithms – previously not available – and to operate on ACE's list, set and string constructs.

To avoid a possible misunderstanding, this is not a venture of ACE/RACE into the field of mathematics per se, as realised in the project Naproche[3], or as outlined in this report[4].

Section 2 of this paper recalls general features of RACE. Section 3 presents the mathematical extension. Section 4 motivates and introduces the functional extension. Section 5 concludes with a summary of the presented extensions and with a discussion of their strengths and limitations.

2 General Features of RACE

For the convenience of the reader and to make this paper self-contained the material of this section is partially copied from (Fuchs 2012).

RACE has the following general features:

- RACE offers consistency checking, textual entailment and query answering of ACE texts.
- RACE does not presuppose knowledge of formal logic or theorem proving, does not require users to understand RACE's internal workings, nor does it require users to control the reasoning process.
- All input of RACE is in ACE, all output is in ACE and English.
- Consistency checking: For inconsistent

[1] http://attempto.ifi.uzh.ch/
[2] http://attempto.ifi.uzh.ch/race/
[3] https://korpora-exp.zim.uni-duisburg-essen.de/naproche
[4] https://jiggerwit.files.wordpress.com/2019/06/header.pdf

ACE axioms RACE will list all minimal inconsistent subsets of the axioms.

- Textual entailment and query answering: If the ACE axioms entail the ACE theorems, respectively ACE queries, RACE will list all minimal subsets of the axioms that entail the theorems, respectively queries. Furthermore, there will be substitutions for all occurring query words.
- RACE uses about 100 auxiliary axioms – not expressed in ACE, but in Prolog using ACE's internal representation – to provide domain-independent general knowledge. In spite of their large number, auxiliary axioms have little impact on RACE's performance since they are only called individually and only when needed.
- RACE is implemented as a set of Prolog programs that can be used locally. Furthermore, RACE can be accessed remotely via its web-client[5] or via its web-service[6].

3 Reasoning with Arithmetic, Linear Equations and Quadratic Equations

Reasoning with positive integers that occur as determiners (*2 apples*, *at most 3 apples*) and with positive integers and reals that occur in measurement nouns (*2.25 l of water*) was described in a previous system description (Fuchs, 2012). Not covered, however, was until now reasoning with ACE's arithmetical constructs, which is part of the present system description.

Here is a brief summary of ACE's arithmetical constructs.

ACE offers numbers that syntactically act as nouns. Numbers are positive and negative integers and positive and negative reals. Furthermore, there are arithmetic expressions (*(X ^ 3) ^ 1/2 – 4*Pi*) built with the help of the operators +, –, *, /, ^ from numbers, variables, proper names and parenthesised subexpressions. Arithmetic expressions can evaluate to numbers and thus count as nouns.

ACE's boolean formulas (*X >= 13.4 and X < 20.*) are built from numbers, arithmetic expressions, proper names and variables with the help of the comparison operators =[7], \=, >, >=, < and =<. Boolean formulas syntactically act as sentences.

When reasoning with arithmetic expressions and formulas one encounters four new phenomena.

- While previously RACE fundamentally relied on unification, i.e. on the syntactic matching of logical atoms, numerical expressions – like those in the formula *100/50 + 8 = 4 + 6* – cannot simply be unified but must be numerically evaluated before being tested.
- While previously the order of processing did not matter, the evaluation of expressions – as in *A is B + C. C is D - 1. B is 2. D is 3.* – must be delayed until all constituents have a value.
- Even after evaluation remain problems of relating formulas, as can be seen in the deduction *X=1 |– X>0.*
- As in standard logic, arithmetical contradictions can involve negation, as for instance in *A is 1. A is not 1.* But there are new forms of contradictions not involving negation, for example *A is 1. A is 2.* or simply *1=2.*

While it is possible to solve the problems associated with these phenomena in RACE's implementation language SWI Prolog, this would amount to duplicating functionality that is available off-the-shelf in the form of mathematical frameworks. Because it efficiently copes with the four phenomena above, because it does not require changes of ACE's syntax, because it is highly efficient, and because of its simple integration, RACE uses SWI Prolog's library clpqr[8] that provides constraint logic programming over rationals and reals, and also logical entailment.

Following are three simple examples that show a range of possible applications of RACE's mathematical extension.

A Banking Problem. A capital *C* is invested in a bank at an interest rate *I* for the duration of *D* years while the bank charges a yearly fee *F*. Then the approximate final balance is

*C *(1+I)^D - F*(1+I)^(D - 1) - F(1+I)*(D - 1)*

disregarding higher powers of *I* in the last term to get a closed expression. This leads to a small

[5] http://attempto.ifi.uzh.ch/race/

[6] http://attempto.ifi.uzh.ch/ws/race/racews.perl

[7] Note: RACE accepts the copula *is* as a synonym for the comparison operator =.

[8] https://www.swi-prolog.org/man/clpqr.html

error that will be ignored here. Given the ACE axioms

The capital C is 1000.00.
The interest I is 0.005.
The yearly fee F is 12.00.
The duration D is 10.
*The balance of the account is C *(1+I)^D - F*(1+I)^(D - 1) - F*(1+I)*(D - 1).*

and the ACE query

What is the balance of the account?

RACE arrives at the balance of 930.05.

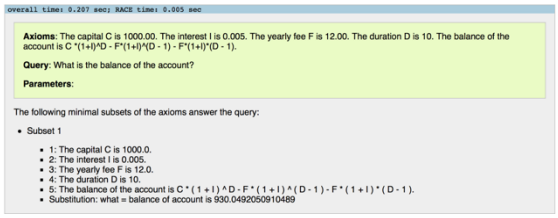


Figure 1: A Banking Problem

The result (Figure 1) is presented as a screenshot of the output window of RACE's web-interface. This window contains the ACE axioms, the ACE query, the subset of the axioms needed to answer the query and the substitution of the query word *what*, i.e. the actual numerical result. The entry "parameters" is used for testing.

A Word Problem. Many word problems are dressed-up arithmetic problems. Here is an example:

A farmer has some cows and some ducks. Altogether he has 100 animals with 260 feet. How many cows and how many ducks does the farmer have?

To solve this word problem, one needs some background information, namely

Every cow is an animal. Every duck is an animal. No cow is a duck. Every cow has 4 feet. Every duck has 2 feet.

Even with this background information the problem cannot yet be solved because a problem-solving strategy is needed. Schwitter (2012) demonstrated how such a strategy could be devised for the Marathon Puzzle that should determine the arrival order of a group of runners. Schwitter's strategy consists of formulating the puzzle in the controlled natural language PENG, and then translating the PENG text into an answer set program (ASP) that is submitted to an ASP solver. Though Schwitter's strategy is elegant and efficient, it is specific to the Marathon Puzzle and cannot be immediately generalised.

Instead, I suggest for word problems that are hidden arithmetic problems a strategy that is perhaps less elegant, but efficient and more general, namely to manually derive from the text the – often linear – equations, thereby taking into account the explicit or implicit background information.

Here is the ACE version of the farmer-cow-duck problem, expressed as three axioms and two queries. Note that the axioms implicitly incorporate the complete background information.

A farmer has a number X of some cows and has a number Y of some ducks.

X+Y=100.

*4*X+2*Y=260.*

/-

What is a number of some cows? What is a number of some ducks?

Submitting these axioms and queries to RACE we get the result (Figure 2) that the farmer has 70 ducks and 30 cows. Note that by design the two queries are answered separately since there are different substitutions of the two occurrences of the query word *what*.

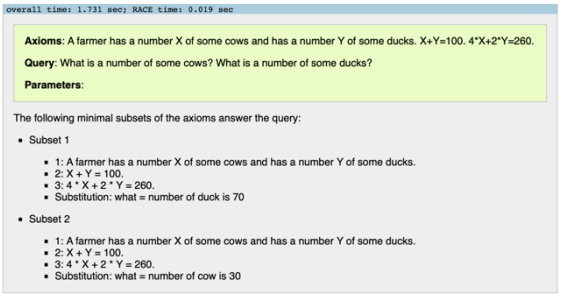


Figure 2: Farmer, Cows and Ducks

Quadratic Equations. Quadratic equations occur in many problems[9], specifically in physics and geometry.

[9] https://en.wikipedia.org/wiki/Quadratic_equation

Unfortunately, clpqr cannot solve quadratic equations directly. However, replacing the quadratic equation *X^2 + P * X + Q = 0* by its two solutions *X = -P/2 ± √ (P^2/4 - Q)* eliminates this stumbling block.

As an example, here is the quadratic equation for the ubiquitous Golden Ratio[10]:

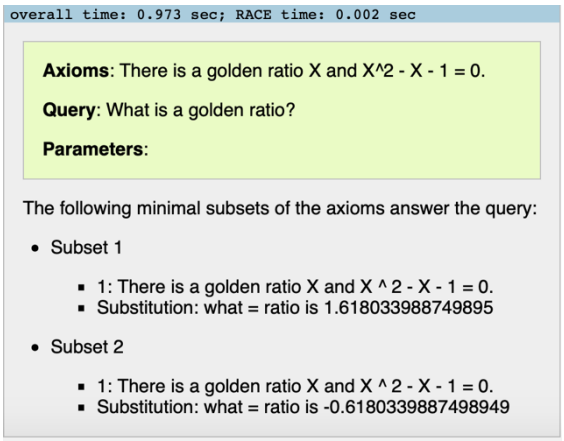


Figure 1: Quadratic Equation for Golden Ratio

Like most quadratic equations this one has two solutions (Figure 3). Only the positive solution pertains to the golden ratio.

The current implementation for quadratic equations has the following syntactic restrictions: The quadratic term has no coefficient, the terms P and Q are integers, integer fractions or reals. Since ACE does not know complex numbers, quadratic equations with complex solutions are flagged with an error message.

Replacing quadratic equations by their solutions solves an important problem, yet in a way that cannot easily be generalised. A mathematical framework more powerful than clpqr could possibly avoid this problem.

4 Extending RACE by a Functional Notation

For most of its intended applications ACE does not need a functional notation, and hence does not provide one. RACE, however, needs a functional notation to extend its reasoning capabilities. Exploiting ACE's list structure, I devised a functional notation in the form *["functor", argument1, argument2, ...]* that allows RACE to call the Prolog predicate *functor(argument1, argument2, ...)*. Note that the functor is expressed as a string to be accepted by the ACE parser without having an entry in ACE's lexicon.

This simple extension has three beneficial consequences. RACE can

- call any built-in or user-defined Prolog predicate,
- make use of recursive algorithms that cannot be implemented otherwise,
- operate on ACE's list, set and string constructs.

Here is a simple example of list operations using three built-in Prolog predicates combined with RACE's arithmetic:

There is a list L of ["append", [1,2,3], [4,5,6], L]
and there is a maximum M1 of ["max_list", L, M1]
and there is a minimum M2 of ["min_list", L, M2]
and there is a result R and R = M1 + M2.

/-

What is a result?

Submitting this example to RACE we get 7 as the result (Figure 4).

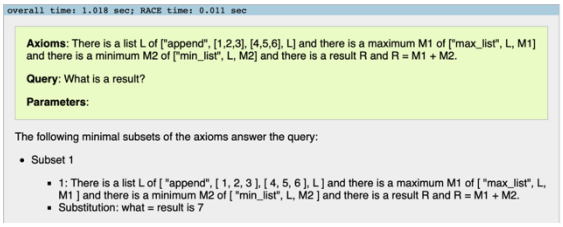


Figure 4: Using Prolog's List Predicates

Figure 5 shows an example of the user-defined Prolog predicate *gcd/3* that implements Euclid's recursive algorithm for the greatest common divisor. In this case a function name is explicitly introduced.

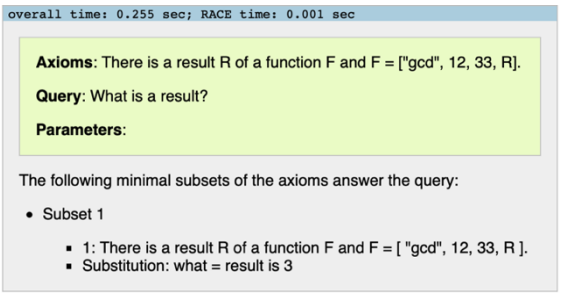


Figure 5: Euclid's Recursive GCD Algorithm

[10] https://en.wikipedia.org/wiki/Golden_ratio

Besides the three beneficial consequences listed above, the functional notation has another practical application. Understanding Prolog programs can pose problems for people not familiar with this language, as it happened recently in the context of a psychology project (private communication, 2021). The psychologist in question had developed a set of Prolog programs to solve statistical problems, but could not help to notice the lack of understanding of his colleagues. The functional notation would have enabled him to incorporate calls to these Prolog programs into ACE texts that his colleagues could understand without knowing much of Prolog.

5 Conclusions

I added to the Attempto Reasoner RACE mathematical and functional extensions and illustrated them by small examples.

RACE's mathematical extension does not raise any questions besides being limited to some extent by the restrictions of the chosen mathematical framework clpqr.

RACE's functional extension, however, raises two issues. First issue: Though the list notation is accepted ACE syntax, the explicit calls of Prolog predicates in an ACE text could be considered a violation of the intention of the Attempto project to hide formality from its users. Second issue: RACE relies on auxiliary Prolog axioms that add domain-independent general knowledge to the domain-specific knowledge of the given ACE axioms (Fuchs, 2012). This reliance on Prolog is increased by the functional extension that allows us to directly call Prolog predicates. Since Prolog has the power of the Turing machine, RACE could in principle deduce any conclusion from the axioms. As a consequence, besides the usual questions of the correctness and the completeness of the reasoning process a further question arises, namely the relevance. What should RACE actually deduce? For instance, RACE's auxiliary axioms enable the deduction *John's cat purrs.* |- *John has a cat.* Is this deduction – that has no logical justification, but is based solely on common sense – acceptable? The answer depends not only on the application domain, but also on the expectations and intuitions of the users, and this – as my experience has amply shown – may be highly debatable.

RACE now covers all language constructs of ACE with the exception of imperative sentences that do not play a role in logical deduction, and two constructs that have no – or at least no generally accepted – logical representation, namely the unconventional modal operators for recommendation (*should*) and admissibility (*may*) that were provisionally introduced into ACE to cover the medical jargon of Clinical Practice Guidelines (Shiffman et al., 2009).

Acknowledgments

Many thanks go to my colleague Uta Schwertel for her essential contributions in the initial phase of the development of RACE. The constructive comments of the reviewers of a previous version of this paper are gratefully accepted. Finally, I would like to thank the Department of Computational Linguistics, University of Zurich, for its hospitality.

References

Norbert E. Fuchs. 2012. *First-Order Reasoning for Attempto Controlled English.* In *Proc. of the Second International Workshop on Controlled Natural Language (CNL 2010)*. Marettimo, Italy.

Norbert E. Fuchs. 2016. *Reasoning in Attempto Controlled English: Non-monotonicity.* In *Proc. of the Fifth International Workshop on Controlled Natural Language (CNL 2016)*. Aberdeen, UK.

Richard N. Shiffman, George Michel, Michael Krauthammer, Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. 2009. *Writing Clinical Practice Guidelines in Controlled Natural Language*. In *Proc. of the Workshop on Controlled Natural Language (CNL 2009),* Marettimo, Italy.

Rolf Schwitter. 2012. *Answer Set Programming via Controlled Natural Language Processing*. In *Proc. Third International Workshop on Controlled Natural Language (CNL 2012)*. Marettimo, Italy.

A quality evaluation framework for a CNL for agile law execution

Ilona Wilmont
HAN University of Applied Science,
P.O. Box 2217, 6802 CE, Arnhem,
The Netherlands
Radboud University Nijmegen,
P.O. Box 9010, 6500 GL, Nijmegen,
The Netherlands
ilona.wilmont@han.nl

Diederik Dulfer
Dutch Tax Administration,
John F. Kennedylaan 8,
7314 PS, Apeldoorn,
The Netherlands
dph.dulfer@belastingdienst.nl

Jan Hof
Dutch Tax Administration,
John F. Kennedylaan 8,
7314 PS, Apeldoorn,
The Netherlands
ja.hof@belastingdienst.nl

Mischa Corsius
HAN University of Applied Science,
P.O. Box 2217, 6802 CE, Arnhem,
The Netherlands
mischa.corsius@han.nl

Stijn Hoppenbrouwers
HAN University of Applied Science,
P.O. Box 2217, 6802 CE, Arnhem,
The Netherlands
Radboud University Nijmegen,
P.O. Box 9010, 6500 GL, Nijmegen,
The Netherlands
stijn.hoppenbrouwers@han.nl

Abstract

The Dutch Tax Administration has developed and exploited a CNL, RegelSpraak, to automate law execution. This CNL is meant to be comprehensible for legal specialists, IT developers and computers. However, quality assessment of rule patterns and their ability to express all relevant tax laws is currently not possible. In this study we evaluate potential quality criteria that offer the capability to evaluate rule pattern quality based on semantic expressive power, cognitive usability and functional and structural correctness. We design a quality framework based on insights from literature review, interviews and observations, which has been qualitatively operationalized. Initial results suggest that they touch on relevant variables, but that further quantitative operationalization and testing of the framework's usability in practice are needed.

1 Introduction

RegelSpraak is a Dutch controlled natural language (CNL) developed in 2008 by the Dutch Tax Administration (DTA), which has since been employed for the purpose of law execution. It is meant to be comprehensible for legal specialists, IT developers and computers alike, offering a common ground for unambiguous communication about legislation.

RegelSpraak is based on RuleSpeak (RuleSpeak, 2021) and has a strong pragmatic focus. It is currently being employed in several projects for the specification and implementation of tax rules, such as income tax, corporation tax and wage tax. For these selected purposes, RegelSpraak functions adequately. However, it is not possible to assess whether RegelSpraak, given its current state, is capable of expressing all relevant tax laws. Nevertheless, new patterns for the formulation of rules are being developed as deemed necessary. The DTA is therefore seeking to develop a framework

for evaluation of rule pattern quality which can be directly used during the process of rule pattern development rather than afterwards. In addition to rule pattern assessment, the framework must be able to assess the quality of the CNL as a whole.

This study focuses on the aspects of and perspectives on quality which may be relevant to assess in the context of a CNL aimed at automated execution of rules. We evaluate potential criteria for assessment of the understandability of RegelSpraak and the quality of a rule pattern, as well as whether additional quality measures are needed. These insights can further the development of rule patterns, as well as the tooling used to implement them.

1.1 The basic structure of RegelSpraak

Two categories of CNLs are those improving readability for human users, and those facilitating an automated semantic analysis (Tommila and Pakonen, 2014). RegelSpraak falls into both categories. The foundations of RegelSpraak lie in the Dutch version of RuleSpeak (RuleSpeak, 2021): a set of guidelines to formulate business rules in a precise yet user-friendly manner. As opposed to RuleSpeak, though, RegelSpraak has a predefined syntax and semantics. The syntax is enforced though the defined language patterns, which have a precisely defined semantics. The use of examples of the intended rule behaviors ensure that the rule analyst understands the implication of the rules specified. These language patterns facilitate the automated execution of the rules, which is a major step forward for the DTA's IT implementation process.

Table 1 presents the types of rules RegelSpraak employs. A RegelSpraak rule always has the following format: [RESULT] IF [CONDITION(S)]. A condition compares attributes, which can have boolean or numerical values, or be dates, enumerations or roles. The result is executed as a consequence of the successful evaluation of the conditions. The results and conditions are connected using carefully composed Dutch phrases to maximize the resemblance to a natural sentence.

The following fictitious example is a RegelSpraak calculation rule. The results part is specified before the listing of the conditions.

```
Rule tax based on travel time
  Valid from 2011 to 2015

    The tax on a train journey must be
    calculated as the TAX_PERCENTAGE of
    (900 minus the travel time by train
    in minutes of the taxed journey)
    if the journey meets all of the
    following conditions:

      - the type of the travel tax is
        equal to 'gross tax based on
        travel time'
      - the travel time by train in
        minutes of the taxed journey is
        larger than or equal to 600
      - the travel time by train in
        minutes of the taxed journey is
        smaller than 900.
```

The aforementioned rule is an instantiation of the following rule pattern:

```
Rule <description>
 Valid from <year> to <year>

   The <attribute> of an <object> must
   be calculated as the <expression>

   if the <object> meets <all | one
   | at least n | at most n | none
   | exactly n> of the following
   conditions:

     - <condition>
     - <condition>.
```

1.2 Two cognitive perspectives on rule authoring

Rule authoring is a cognitively demanding task for analysts, and it is therefore important to review some cognitive process requirements. Analogies from modeling and writing research are used (Wilmont, 2020).

Early research on story writing found that writers use three strongly intertwined processes: *planning*, *sentence generation*, and *revision* (Hayes and Flower, 1986). Furthermore, writing is goal directed and goals are hierarchically organized. Skilled writers comment on their goals early in the process and continue to define subgoals after having determined the main goal. This process can also be observed in conceptual and scientific modeling (Ross et al., 1975; Sins et al., 2005). The ability to abstract the essential meaning from concrete activities or observations, as well as the ability to engage in metacognitive monitoring behavior, are important success factors in modeling (Wilmont et al., 2013). Tax rules can be viewed as a model of the legal reality, therefore the importance of these skills applies to rule authoring as well.

Following Gemino and Wand (2003), during rule authoring an analyst can take on one of two perspectives: *reader* or *writer*. The rule writer is the active role, responsible for translating knowledge into

Rule type	Description
Decision rule	Deduces a true/false result
Calculation rule	Calculates a number or amount
Time/date rule	Calculates dates
Consistency rule	Validates whether the value of an attribute is consistent when compared to the value of another attribute, or a validity check for one attribute
Assigning characteristics	Determines the value of a characteristic
Initialisation rule	Assigns a value to an attribute if the value is not being determined by a rule or an input message
Object creation rule	Creates a new object
Fact creation rule	Creates a new fact

Table 1: Types of RegelSpraak rules

rules. On the contrary, the rule reader is passive, concerned with reading the rules and constructing a mental model of the domain based on the information contained in the selected rules, RegelSpraak's grammatical structure and internalized knowledge gained from prior experiences.

This dichotomy translates perfectly to the roles assumed in rule development teams. Typically, they are made up of tax experts and rule analysts. Tax experts have highly specialized knowledge on legal contexts in which the rules are meant to function, as opposed to rule analysts who have intricate knowledge of RegelSpraak syntax and methods for rule authoring. Therefore, tax experts are primarily concerned with reviewing what the rule analysts have created. However, despite the fact that each team member is likely to operate in a dominant role, everyone will inevitably have to switch between the roles occasionally during the process of rule authoring. It is therefore essential that the quality assessment framework flexibly accommodates reflection on team progress from both the reader and writer perspectives.

1.3 Evaluating CNL quality

As RegelSpraak is constantly evolving, fast and targeted quality evaluation during development is crucial. Interestingly, earlier work on the evaluation of CNLs has focused mostly on the usability aspects, in particular *understandability* and *learnability* (Kuhn, 2009). However, during interpretation tasks it proved difficult to determine whether people truly understand the CNL or whether they follow syntactical patterns to complete the task. Kuhn (2009) demonstrated that graphical representations, which prompted people to reason about the meaning of statements, encouraged understanding. However, RegelSpraak projects may contain over 1000 rules, in which case graphical representations would not be suitable. Additionally, basic knowledge of logic helped to evaluate statements in which potentially ambiguous constructs such as an inclusive OR are used. This implies that for the user, some background knowledge is desirable.

Concerning RegelSpraak itself, a set of rules can be viewed as a model of legal reality. From this point of view, conceptual modeling quality criteria can be applied for evaluation. An early quality framework differentiated between syntactic, semantic and pragmatic quality (Lindland et al., 1994). This framework has been expanded and revised to become the SEQUAL (SEmiotic QUALity) framework for model evaluation (Krogstie et al., 2006), which provides theoretical, qualitative guidelines. It encompasses the following aspects: physical representation and availability, empirical layout and readability, syntactic requirements, semantic understandability, pragmatic impact of the activated model, socially mediated shared understanding and organizational aspects such as which parts of the model can be changed.

For our purposes the original three aspects are leading. Syntactic quality focuses on how to enhance the formal syntax, to implement error detection, error prevention and recovery. Semantic quality refers to the human interpretation of the model, the relation between one's knowledge and the model. Pragmatic quality concerns model activation; that is, how the models are interpreted by both social and technical actors. Technical interpretation demands complete models with operational semantics, whereas social interpretation requires flexibility and effective abstraction. We use the aspects from the revised SEQUAL version to guide the development of the RegelSpraak evaluation criteria.

2 Methodology

For this exploratory project, we organized our activities in three phases: 1) an exploration of potential quality criteria, 2) an investigation of the criteria in terms of user perspectives and cognitive processes, and 3) integration into a conceptual framework.

The exploration of quality criteria was done through a literature review, which included both scientific and grey literature. The domains of CNL, software quality and linguistic quality were queried. Additionally, documentation provided by the DTA reviewed.

For the exploration of user perspectives, we conducted interviews with five DTA employees who worked in varying functions. The sample consisted of one developer and four rule analysts, of whom one had a legal background and one was responsible for RegelSpraak training courses. We set up an interview guide to include the following aspects: generic linguistic quality aspects, the role of RegelSpraak in the daily workflow, difficulties and best practices when using rules and patterns, incentives for new pattern development, the documentation of design decisions and how best to teach it to new employees.

Interviews were audio-recorded with the participants' consent, and analysed directly from the audio files. We denoted timestamps and took elaborate notes whenever utterances of the following types were encountered:

- Decision-making aspects of determining rule and pattern quality,
- Underlying principles and frameworks which analysts and developers (unconsciously) use to test and validate new rules and patterns during development,
- Explicit mentions of routine actions.

Furthermore, we conducted a semi-structured, non-participant observation of a rule pattern design session, in which three rule analysts, one developer and a project leader took part. This session was held via Cisco Webex, and the researchers switched off their audio and cameras to appear as unobtrusive as possible. The session was video-recorded with the participants' consent. The following structural framework was used to analyse the video:

- Spontaneous mention of quality criteria (either implicit or explicit),
- Differences in interpretation of quality criteria,
- Mention of operational norms for quality criteria,
- Allocation of attention to different quality criteria,
- The use of the human versus the rule engine perspective in the discussion,
- The process of decision making in the context of differing opinions,
- Quality aspects which might block further pattern development progress.

The final integration phase took place in short design cycles in which DTA employees were actively involved. Significantly recurring quality criteria were deduced by triangulating the literature review, interview and observation results. The integration process focused on formulating operational criteria for each of the quality criteria, and on designing a user-friendly procedure for testing rule pattern quality in daily practice.

It should be noted that the primary scope for this initial phase of the RegelSpraak evaluation project is focused on standardizing the best practices to align understandability for users and formal syntax as produced by rule analysts. A formal, computational evaluation of the resulting RegelSpraak rules is beyond the scope of this paper.

3 Results

Our multi-faceted exploration yielded insights from literature review, interviews, observations and active design cycles. The final set of quality criteria that we selected consisted of the following: Readability, Recognizability, Explainability, Usability, Completeness, Syntax, Complexity, Efficiency, Compositionality, Domain Independence, Technical Executability. These criteria were then differentiated and described from both a *fiscal* and a *computational* perspective. The resulting categorizations are shown in Section 3.3: the Fiscal perspective is shown in Table 2, and the Computational perspective in Table 3.

We continue to describe how the iterative research process resulted in the design of our framework. Firstly, the results of our quality criteria exploration are discussed, followed by the integration

with user perspectives and finally the translation to actual framework development.

3.1 Phase 1: An exploration of quality criteria

We approached the exploration of quality criteria from four perspectives: linguistics, conceptual modeling, software engineering and usability. Major themes that emerged were semantic expressiveness, cognitive usability and the functional and structural aspects associated with formal languages.

3.1.1 Semantic expressiveness

As described in Section 1.3, syntax, semantics and pragmatics form the basis for the framework. At the intersection of linguistics, conceptual modeling and usability, the ability to express the desired semantics in a rule pattern is one of the most crucial themes. Ultimately, if the model does not convey the right message to the stakeholders for whom it was created, the relevance of all other aspects of model quality may be questioned. In the case of RegelSpraak, semantic consensus between fiscal experts, rule analysts and IT developers must be achieved.

Legal texts are often formulated with precision and disambiguation in mind, yet many linguistic constructs are used to express desired nuances. Despite RegelSpraak's user-centred focus, its main goal is to automate rules, for which human texts have to be transformed to fit into a formal syntax. This requires a delicate balance between the abstraction levels of both rule patterns and rules, as they must be designed to express as many diverse meanings as possible yet not become overly generic.

Guidelines for how to incorporate maximal expressive power in a formal rule pattern can be found in Grice's maxims (Grice, 1991). In short, Grice's maxims dictate that contributions to a conversation should be maximally informative within minimal wording, contain no untruths, be precise, unambiguous and above all relevant. Whenever there is a possibility that ambiguity might occur, for instance with 'can'-patterns, they must be tested extensively against a diverse sample of concrete rules until there is no room left for interpretation. This records how fiscal concepts and norms are interpreted and applied. From this test, essential relevance and pattern complexity minimization can be determined.

Furthermore, automated rule engines can only deal with a specified vocabulary, which further emphasizes the importance of a concise vocabulary. If patterns are designed to accommodate this vocabulary and structured in such a way that chosen signalling words allow minimal freedom of interpretation, RegelSpraak forms an excellent basis for facilitating flexible, or agile, law execution.

3.1.2 Cognitive usability

Cognitive usability requirements featured prominently in the interviews. The concepts of *understandability*, *readability* and *explainability* were considered to have the highest priority, although the interview results suggest that their precise definitions and relations to each other are all but clear-cut.

Understandability is required in different forms. For one, rule analysts are taught the existing rule patterns and what they can express. It is therefore essential that the logic and abstraction associated with rule patterns are understandable. For another, in the current situation fiscal experts do not write rules. They only read what the analysts have written, and listen to analysts explaining their creations. Therefore, readability and explainability may be seen as critical understandability prerequisites.

When the interviewees were prompted to comment on a rule pattern the researchers considered poorly understandable, aspects such as sentence complexity, poor abstraction (*"too much detail in one construct", "using a summation or a basic calculation should be coverable with 'calculation', and the context should clarify how the calculation is to be performed"*), structure (*"diverse options have been written out in different places in the pattern"*) and clarity of option meanings (*"the options in this pattern raise too many questions. For example, why is 'first/last' here? Is this variable the first of a sequence?"*) were mentioned. From this, we deduce that structure and abstraction level are critical understandability requirements for the rule pattern. For the terminology used in the rule pattern, unambiguous interpretation is most important.

One of the interviewees suggested the following: *"it would be good to provide such generic patterns with instantiated examples, but because of a combination of priority and time this unfortunately does not happen well enough"*. Research confirms that modelers use abstraction skills to actively connect generic concepts to concrete activities and objects through generalization and instantiation (Theodor-

akis et al., 1999). This is therefore another critical aspect contributing to understandability to be included in the quality evaluation framework.

3.1.3 Functional and structural evaluation

RegelSpraak has many parallels with programming languages. For functional and structural assessment, inspiration may thus be taken from existing software quality criteria, such as the ISO/IEC 25010 (International Organization for Standardization, 2017). Two perspectives are relevant: *functional quality*, which measures the extent to which the software satisfies the functional requirements for its intended use, and *structural quality*, which measures the extent to which the software satisfies non-functional requirements such as robustness, usability, maintainability, compositionality or complexity. Functional assessment must be considered during creation of rule patterns, whereas structural assessment must be performed immediately after creation. It is thereby important to assess the current state of a rule or rule pattern against the desired target state, using both objective data and user perceptions, and describe the results both in terms of qualitative descriptions and hard numbers (Paroubek et al., 2007). Examples should be used to the point of saturation to ensure the stability of the rule patterns in terms of grammar and expressiveness (Veizaga et al., 2020).

One interviewee made it particularly clear that the principle of compositionality is essential for linking the structure of a rule pattern to the way it is to be implemented in the rule engine. In the linguistic sense, it concerns the building blocks of sentences and the relations between them, and in the computational sense that a system should be designed by linking smaller, independent subsystems so that recursive reasoning about the meaning and workings of the system is possible. The rule pattern - implementation connection is made by relating the modular, atomic building blocks of a law as simple and consistently as possible to derive equally modular rule patterns from them, which can thereafter be implemented as such.

3.2 Phase 2: User perspectives and cognitive processes

In order to be able to design quality criteria, insights in the current way of rule pattern use and development are necessary. Additionally, clarity on the prospective users is needed. From the interviews and observations we gained some insights in how people *think* they use rule patterns, how they are being used in a collaborative design session and which future users they envision.

3.2.1 Prospective users

Concerning prospective users, the way in which these opinions were expressed was very much diversified. Whereas one interviewee idealized that "John Doe must be able to understand it", other interviewees were more conservative: "It is still a domain-specific language, so it is of particular importance that the 'inhabitants' of the domain, namely the rule analysts and the fiscal experts, are being served."

In the current system, rule analysts are being trained in what can be expressed with existing rule patterns. There are no fiscal experts who design rules independently. Fiscal experts are only engaged as 'rule readers'. Note that this concerns only the concrete, implementable rules, not rule patterns. Therefore, the emphasis of quality criteria for fiscal experts focuses only around understandability and readability. There thus exists a rather large divide between the rule analyst and the fiscal expert in the way they work with rules and rule patterns.

3.2.2 Rule pattern development best practices

For the way of working when designing rule patterns, best practices have already been formulated. Rule pattern design is always done in a multidisciplinary team, which undergoes a design cycle consisting of the following phases: Analysis, Specification of rule pattern, Review with designated team, Implementation, Create documentation and Use pattern in practice. The activities of explaining ideas for patterns to colleagues and paying special attention to understandability are emphasized. However, all interviewees agreed that more structure for evaluation is desired. "We don't really have a good framework for testing pattern development, so many people contribute ideas from their own backgrounds and perspectives, we miss a common framework that states what is important and why. We must all work from the same framework."

The most important insights emerging from the work session observation was that several quality criteria such as relevance, consistency, uniformity, flexibility and non-ambiguity were already spontaneously mentioned, and participants also mentioned afterwards that these were not consciously

planned or organized. The human usability perspective featured most prominently, although some operational criteria are discussed. The questions a rule pattern design raises leads the discussion rather than criteria. Additionally, the explicit differentiation between rules and rule pattern was not made, they were being used interchangeably which sometimes led to confusion. Finally, this observation also gives valuable insights in the dynamic, flexible workflow of daily practice, which is an important criterion to keep in mind when designing the evaluation framework to keep it usable.

3.3 Phase 3: Developing a quality assessment framework

Based on the input from the literature review, interviews and observations we have been able to deduce a set of quality criteria for the evaluation of RegelSpraak rules and rule patterns. The main challenge that remains is to operationalize them to the extent that they are precise yet workable.

3.3.1 Differentiating perspectives

As mentioned above, our set of quality criteria consists of the following: Readability, Recognizability, Explainability, Usability, Completeness, Syntax, Complexity, Efficiency, Compositionality, Domain Independence, Technical Executability.

1. Human - Machine After reviewing the set of quality criteria, we came to the conclusion that most can be interpreted from two perspectives: the human and the machine. The human perspective encompasses how fiscal experts and rule analysts work with the rules and patterns, whereas the machine perspective embodies the computational implementation and execution of rules. Readability, Recognizability and Explainability are uniquely human, but the other criteria can be operationalized from multiple perspectives.

2. Read - Write On top of the human - machine differentiation, we included the read - write distinction within each perspective, as mentioned in Section 1.2. From the human perspective, fiscal experts read rules and rule analysts write rules. From the computational perspective, the rule engine reads the rules as input and writes them to executable RegelSpraak rules as output.

3. Rule - Rule Pattern The next critical step was to differentiate explicitly between the two abstraction levels of concrete, implementable rules, and abstract rule patterns still to be instantiated. We had previously observed that the two were being used interchangeably in practice, but for precise operationalization a clear distinction is necessary.

Revision: Fiscal - Computational However, during the consecutive iteration it became clear that simply distinguishing between a human and a computational perspective was too simple. For example, when defining a criterion such as Complexity, even from a computational perspective human developers are responsible for delivering the right level of complexity to the machine. Therefore, what was initially the 'Human' perspective became the 'Fiscal' perspective, and 'Machine' changed to 'Computational'.

Additionally, it became clear that applying the Read - Write and the Rule - Rule Pattern distinctions to the computational perspective resulted in an artificial description that did not correspond with what could be tested in practice. In particular, the implementation and execution of instantiated rules could be seen as a result of successful rule pattern instantiation, and therefore not truly relevant. Therefore, we unified the Computational perspective to focus on the Rule pattern level and drop the Read - Write distinction. The core focus then becomes the implementability of the rule pattern, whereby 'implementable' is viewed from the usability perspective of the UX designer as creating a usable interface for the rule engine, and from the perspective of the developer as delivering efficient and complete code.

The final differentiation of perspectives and categorization of criteria within the Fiscal perspective is shown in Table 2, and the Computational perspective in Table 3.

3.3.2 Operationalizing qualitative norms

Tables 2 and 3 show that certain criteria are unique to a perspective, whereas others are needed from different perspectives. The most pervasive one is Complexity. One could argue that within the Fiscal - Read perspective, complexity is integrated in all criteria, since an overload of complexity hinders readability, recognizability and understandability. For the Fiscal - Write and Computational perspectives, however, complexity needs to be specifically defined. We began the operationalization with qualitatively formulated norms, touching on core variables. From the Fiscal perspective, Complexity of a rule or rule pattern encompasses the following four

	Fiscal - Rule Pattern	Fiscal - Rule
Read	Recognizability Readability Explainability	Recognizability Readability Explainability
Write	Complexity Completeness Usability Syntax Compositionality Domain Independence	Complexity Completeness Usability Syntax

Table 2: The Fiscal perspective.

Computational - Rule Pattern
Complexity
Completeness
Usability
Compositionality
Efficiency

Table 3: The Computational perspective.

elements: Number of variables, Number of variable types, Number of relations between variables, and Order. In case of an instantiated rule, it can be made up of several pieces of rule pattern. Therefore it is important to minimize for each rule the scope and thereby the number of rule patterns or pattern parts that have to be used to express the meaning of the rule. From the Computational perspective, however, the non-functional requirements of Performance efficiency and Maintainability were the critical variables. The three qualitative definitions of Complexity are as follows:

Complexity 'Fiscal - Write - Rule Pattern' A rule pattern must be divided into sub-patterns if the following complexity conditions cannot be met:

1. Is the number of variables minimized?
2. Is the number of operators minimized?
3. Are the types of values a variable can take minimized?
4. Is the number of relations between variables minimized?
5. Is the scope of the pattern minimized?
6. Are the variables presented in a logical order in which consecutive variables build on prior variables to maximize readability, usability and domain independence?

Complexity 'Fiscal - Write - Rule' A rule must be divided into sub-rules or enumerations if the following complexity conditions cannot be met:

1. Does the number of legal concepts represented in the rule remain such that connected sentence parts remain readable, recognizable and understandable?
2. Is the number of operators in the rule minimized?
3. Is the number of legal concepts in the rule minimized?
4. Is the number of relations between legal concepts and (parts of) rule patterns minimized?
5. Is the scope of the rule minimized?
6. Have the legal concepts been defined on the highest possible level of abstraction such that the semantics are still representative of the law?
7. Is legal maintenance of rules easy to perform?

Complexity 'Computational' Verify whether the complexity of the code to be implemented is such that performance efficiency and maintainability are guaranteed:

1. Is the number of functions needed to implement the rule pattern minimized?
2. Is the number of steps needed to go from the first to the final step in the function minimized?
3. Do the functions have a modular structure in relation to each other?
4. Is there legacy code that is no longer functional?
5. Which critical consequences may happen if a rule pattern fails? How many steps are needed to deduce this?
6. Which non-critical consequences may happen if a rule pattern fails? How many steps are needed to deduce this?
7. How does error handling in the rule pattern implementations take place?

8. Can rule pattern maintenance be efficiently and accurately performed?

4 Discussion and future work

Given the current framework for relevant quality criteria and the positioning of these within clearly delineated perspectives, a rough standard has been created to improve insights in the quality of Regel-Spraak rule patterns and stimulating critical discussions.

As the study progressed, the question arose to which extent rule pattern quality could be studied separately from the process of rule creation in practice. One of the best ways to test the meaning of a rule pattern remains to instantiate it with existing legal procedures. This led us to further emphasize the importance of making the Rule - Rule pattern distinction, and the Read - Write distinction on top of it. A usable framework with impact will therefore always have to combine analytic evaluation based on abstract quality criteria with user evaluation based on actual implementation tests, preferably in an iterative, flexible fashion.

We have not yet been able to test the criteria in the practice of rule pattern design. However, an initial qualitative review of the criteria suggested that the content indeed touches on relevant variables. The most important next steps are to further refine the best practice cycle the DTA team has already set up to incorporate the evaluative criteria, to focus specifically on how to implement them so that they are usable within the flexible dynamic of design sessions, and to determine the level of quantitative operationalization needed. All these factors need to be incorporated in practical usability testing. Further quantitative validation of our design choices to demonstrate that they are representative of legal constructs need be performed, for example using text mining or surveying a large population of legal professionals.

For usability in practice, the core procedure boils down to awareness of within which predefined perspective a task at hand can be classified, to take note of the criteria defined within that perspective, and to use the operational qualitative criteria to evaluate the result, within the dynamic team discussion. A trade-off will have to be made between the purposes of stimulating critical discussion about rule pattern proposals, and actually measuring quality in a more quantitative sense in relation to a predefined quality standard. It will be interesting to study what numerical constraints on understandability may be useful, such as incorporating the known limits of human working memory (3-5 items) (McCutchen, 1996; Ricker et al., 2010) to constrain the number of nested elements, operators and parameters.

References

A. Gemino and Y. Wand. 2003. Evaluating Modeling Techniques Based on Models of Learning. *Communications of the ACM*, 46(10):79–84.

Paul Grice. 1991. *Studies in the Way of Words*. Harvard University Press.

J.R. Hayes and L.S. Flower. 1986. Writing research and the writer. *American Psychologist*, 41(10):1106–1113.

International Organization for Standardization. 2017. ISO/IEC 25010:2011. https://www.iso.org/standard/35733.html.

John Krogstie, Guttorm Sindre, and Havard Jorgensen. 2006. Process models representing knowledge for action: A revised quality framework. *European Journal of Information Systems*, 15(1):91–102.

Tobias Kuhn. 2009. How to Evaluate Controlled Natural Languages. In *Workshop on Controlled Natural Language*, Marettimo, Italy.

Odd Ivar Lindland, Guttorm Sindre, and Arne Solvberg. 1994. Understanding quality in conceptual modeling. *Software, IEEE*, 11(2):42–49.

Deborah McCutchen. 1996. A capacity theory of writing: Working memory in composition. *Educational Psychology Review*, 8(3):299–325.

Patrick Paroubek, Stéphane Chaudiron, and Lynette Hirschman. 2007. Principles of Evaluation in Natural Language Processing. In *Traitement Automatique Des Langues*, volume 48, pages 7–31.

Timothy J. Ricker, Angela M. AuBuchon, and Nelson Cowan. 2010. Working memory. *Wiley Interdisciplinary Reviews: Cognitive Science*, 1(4):573–585.

D.T. Ross, J.B. Goodenough, and C. A. Irvine. 1975. Software Engineering: Process, Principles, and Goals. *Computer*, 8(5):17–27.

RuleSpeak. 2021. RuleSpeak® —— Let the Business People Speak Rules! http://www.rulespeak.com/en/.

Patrick H. M. Sins, Elwin R. Savelsbergh, and Wouter R. van Joolingen. 2005. The Difficult Process of Scientific Modelling: An analysis of novices' reasoning during computer-based modelling. *International Journal of Science Education*, 27(14):1695–1721.

M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos. 1999. Contextualization as an Abstraction Mechanism for Conceptual Modelling. In *Conceptual Modeling - ER '99, 18th International Conference on Conceptual Modeling, Paris, France, November, 15-18, 1999 Proceedings*, volume 1728 of *Lecture Notes in Computer Science*, pages 475–489. Springer Berlin / Heidelberg.

T. Tommila and A. Pakonen. 2014. Controlled natural language requirements in the design and analysis of safety critical I&C systems. Technical Report VTT-R-01067-14, VTT Technical Research Centre of Finland.

Alvaro Veizaga, Mauricio Alferez, Damiano Torre, Mehrdad Sabetzadeh, and Lionel Briand. 2020. On systematically building a controlled natural language for functional requirements.

I. Wilmont, S. Hengeveld, E. Barendsen, and S. J. B. A. Hoppenbrouwers. 2013. Cognitive Mechanisms of Modelling: How Do People Do It? In *Conceptual Modeling - 32th International Conference, ER 2013, Hong-Kong, China, November 11-13, 2013, Proceedings*, volume 8217 of *Lecture Notes in Computer Science*, pages 74–87. Springer Berlin Heidelberg.

Ilona Wilmont. 2020. *Cognitive Aspects of Conceptual Modelling: Exploring Abstract Reasoning and Executive Control in Modelling Practice*. PhD thesis, Radboud University Nijmegen.

Towards CNL-based verbalization of computational contracts

Inari Listenmaa
Maryam Hanafiah
Regina Cheong
Andreas Källberg
{ilistenmaa, maryammh, reginacheong, akallberg}@smu.edu.sg
Singapore Management University

Abstract

We present a CNL, which is a component of L4, a domain-specific programming language for drafting laws and contracts. Along with formal verification, L4's core functionalities include natural language generation. We present the NLG pipeline and an interactive process for ambiguity resolution.

1 Introduction

We introduce L4, a prototype[1] domain-specific language (DSL) for drafting laws and contracts. L4's applied focus places it within the "Rules as Code" movement (e.g. OpenFisca, Catala (Merigoux et al., 2021)) that itself draws on early computational law thinking (Sergot et al., 1986; Love and Genesereth, 2005). But rather than focusing on encoding laws into existing programming languages, we devise an external DSL designed for legal specification. From this specification, we generate a range of output formats. Key augmentations include IDE support, formal verification engines, transpilation to operational rule engines, and natural language generation (NLG). The latter is done via a CNL implemented in Grammatical Framework (GF, Ranta 2004), and will be the focus of this paper.

Motivation for a DSL The intended user of L4 is not a law firm, but rather an organization or a person who wants to bypass law firms. As a possible early adopter profile, we envision a technology startup drafting their initial agreements in L4. Drafting rules as code enables the owners to keep track of dependencies and detect potential conflicts, without needing a lawyer.

More generally, a formal encoding of rules allows the creation of different end-user applications. When the specification changes, it is only necessary to implement the change in the L4 source, and regenerate the applications.

Motivation for NLG In order to communicate with the rest of the world, the L4 encoding needs to be translated into non-technical end-user products, such as natural language documents and interactive web applications. We present two NLG applications in this paper.

- An expert system, which generates interview questions from L4 code, uses the user input to query an automated reasoner, and presents the reasoner's answers in natural language (Section 3.1).
- In the future we aim to support the generation of an isomorphic natural language version of L4 code (Section 3.2).

In the remainder of the paper, we present a small example of L4 code in Section 2, a brief introduction to the NLG pipeline in Section 3, the CNL in Sections 4 and 5, and finally, future work in Section 6.

2 L4 example

We demonstrate L4 with the following scenario, simplified from an experiment to draft existing regulations in L4 (Morris, 2021). There are certain rules for whether a legal practitioner may accept an appointment in a business. For instance, if the business would share the legal practitioner's fees with unauthorized persons, the legal practitioner is not allowed to accept the appointment.

In Figure 1, we model this rule in L4 (in its current syntax). First, we declare the data types for `Business` and `LegalPractitioner`, and two predicates, `MayAcceptAppointment` and `UnauthorizedSharingFees`. Then, we formulate a rule to say that a legal practitioner may not accept

[1] L4 is a work in progress, and this article presents a snapshot of the project as of June 2021. Any concrete examples of L4 code or the CNL may change in a few months.

```
# types and predicates
class
  Business
  LegalPractitioner

decl
  MayAcceptAppointment
    : LegalPractitioner ->
      Business -> Bool
  UnauthorizedSharingFees
    : Business -> Bool

# rules
rule <no_sharing_fees>
 for b : Business,
     l : LegalPractitioner
  if UnauthorizedSharingFees b
then not MayAcceptAppointment l b
```

Figure 1: L4 code for a rule "legal practitioner may not accept an appointment in a business that involves sharing fees with unauthorized persons"

an appointment in a business which involves sharing fees with unauthorized persons. The current syntax of L4 is adopted from functional programming languages. For example, the type signature for `F : A → B → Bool` means *"the function F takes an A and a B, and returns a Boolean"*.

In the simplest case, we can look up the names of the classes and predicates from any large lexicon: `Business` is just a single noun, and `LegalPractitioner` is either a compositional noun phrase of *legal* (adjective) and *practitioner* (noun), or a multi-word expression in a domain-specific lexicon. The two predicates are more complex, since they involve a full verb phrase. Figure 2 shows an optional lexicon, where the user can write a CNL description of the predicates.

A predicate name like `UnauthorizedSharingFees` is used throughout the program code, and the NLG engine will use the description *involves sharing fees with unauthorized persons*—not as an immutable string, but parsed with a GF grammar into a deep syntax tree. The placeholders, shown in the description of `MayAcceptAppointment`, are optional, if the arguments appear predictably: subject for a unary predicate, and subject and object for a binary predicate. Any CNL description of the form *predicate* is treated as `{Arg1}` *predicate* `{Arg2}`. However, if we wanted to have an argument in another role, then the placeholders are obligatory, as shown in Figure 3.

```
# optional CNL descriptions
lexicon
  UnauthorizedSharingFees
   @ "involves sharing fees with
      unauthorized persons"
  MayAcceptAppointment
   @ "{LegalPractitioner} may
      accept an appointment
      in {Business}"
```

Figure 2: Optional lexicon with CNL description of predicates

3 NLG from L4

L4 is implemented using Alex (Dornan and Jones, 2003) and Happy (Marlow and Gill, 1997).

First, an L4 program is parsed into an L4 abstract syntax tree (AST). That tree is used as an input to all of the output formats named in Section 1. The L4 abstract syntax and the CNL descriptions are used in two NLG applications, which are presented in the remainder of this section.

3.1 Expert system

Listenmaa et al. (2021) presents the expert system in more detail. In this paper, we only introduce what is necessary to set the context for the CNL.

Interview questions The first output target for the expert system is a set of interview questions. Suppose we want to ask a user interactively whether their business may employ a legal practitioner: then we transform every predicate that is a precondition to `MayAcceptAppointment` into a question.

```
decl
  UnauthorizedSharingFees
   : Business
     -> LegalPractitioner
     -> Bool
   @ "{Business} involves sharing
      {LegalPractitioner}'s fees
      with unauthorized persons"
```

Figure 3: Alternative example, where UnauthorizedSharingFees is a binary predicate

1. Is your business incompatible with the dignity of the legal profession?
2. Does your business involve sharing fees with unauthorized persons?
3. ...

The questions are embedded in a Docassemble interview. Docassemble is an open-source legal expert system, where users answer a set of questions through a browser interface. Their responses are then compiled together into a document, or processed further in other applications. In our system, we use the answers to query an automated reasoner.

Reasoner output in natural language The second output target is a verbalization of answers that we get from an automated reasoner. Previously we posed questions to the user—suppose they answered "yes" to question 2. The user input is sent to an automated reasoner, in order to query whether the user may employ a legal practitioner in their company. The reasoner's answer to the query is then rendered in natural language.

> Your business may not employ a legal practitioner, because
>
> - it involves sharing fees with unauthorized persons, and
> - a legal practitioner may not accept an appointment in a business that shares fees with unauthorized persons.

The reasoner we use is s(CASP) (Arias et al., 2018): a logic programming language for Constraint Answer Set Programming. The parameters defined in an L4 source file are used for producing the Docassemble interview and s(CASP) code. Using the user inputs to the interview questions to execute the s(CASP) code, a solution satisfying all the constraints is generated. The solution is then restructured in natural language and becomes the conclusion the user receives.

As the NLG process involves restructuring, this necessitates disambiguation at the input phase. For instance, the transformation of *involves sharing fees* to *a business that shares fees* is only possible if we know that *sharing* is a gerund form of the verb *share*, and not just a noun. The disambiguation is discussed further in Section 5.

Our work differs from approaches such as Schwitter (2012) in that we don't use CNL to formulate a specification or query—we use CNL as means to thoroughly understand the user predicates, so that we can verbalize answers. Our approach is more similar to De Vos et al. (2012), who annotate answer-set programs for verbalization, except that we are not limited to ASP.

3.2 Isomorphic natural language representation

Our future work includes an isomorphic representation of the L4 rules in a natural language as an output target. Suppose our previous rule about unauthorized fee sharing is just one rule among many that dictate whether or not a lawyer is allowed to accept an appointment. Then the isomorphic representation would print out a comprehensive guide on employing legal practitioners—for instance, in a form of a contract to be signed, or as a set of regulations to be published on a web page. If the underlying rules change, then a new document can be generated from an updated L4 source.

On the level of individual rules, we follow Ranta's (2011) translation from logic to natural language: the first step is a literal translation from the programming language abstract syntax to GF abstract syntax, followed by internal transformation of the GF trees to achieve more natural output. However, this work is only in the absolute beginning, and we haven't solved the difficult questions of generating a full NLG pipeline (Reiter and Dale, 2000) to output a complete document from the L4 code.

4 CNL

All of the NLG depends on correct understanding of whatever entities the user defines. Therefore we are using a CNL to restrict user input. In this section, we explain the general principles of the design and implementation, and in Section 5, we explain how ambiguous user input is transformed into CNL variants, with concrete examples.

4.1 Goal of the CNL

Note that the CNL does **not** have formal semantics—L4 as an actual programming language, with its formal verification and transpilation to operational engines, is better suited for that. The CNL is primarily a tool for getting a syntactically precise source for NLG. The default mode of the NLG is a natural-looking, potentially imprecise language. But we envision an option to output syntactically precise CNL as well, if the user so wishes.

4.2 CNL design

The CNL itself is a subset of English language, with disambiguation techniques à la Attempto Controlled English (Fuchs and Schwitter, 1996), ClearTalk (Skuce, 2003) and other CNLs with an unambiguous syntax—see Kuhn (2014) for a survey. The current language features include

- Hyphens for compound nouns: *parking-fee*
- Brackets and word order to show attachment: *[saw with a telescope] the astronomer*
  any [(business, trade or calling) in Bali]
- Disambiguation for past tense. If the fragment has no disambiguating context, we assume: *did prohibit* is past tense, *prohibited* is past participle.

Ambiguity between transitive and intransitive is resolved based on the arity of the predicate. Note that a predicate can be any part of speech, and any part of speech can be transitive or intransitive. Take a word like *clear*, which can be a noun, verb or an adjective. If no CNL description is given, we assume the following:

- `Clear : X → Y → Bool` is a transitive verb, *X clears Y*, and
- `Clear : X → Bool` is an intransitive adjective, *X is clear*.

If we want to use another part of speech or arity, we need to provide a CNL description.

- ```
  Clear : X → Bool
        @ "{X} clears"
  ```
  is an intransitive verb (e.g. "the clouds clear"),
- ```
  Clear : X → Bool
        @ "{X} is in the clear"
  ```
  is an adverbial where *clear* is a noun, and
- ```
  Clear : X → Y → Bool
        @ "{X} is clear to {Y}"
  ```
  is a transitive adjective, which takes its complement with the preposition *to*.

As the project is still in an early stage, the details of the CNL are likely to change. Furthermore, we expect that the CNL features have to be adjusted for every new language. For example, Malay does not differentiate between past tense or past participle, in this case *did prohibit* and *prohibited* both translate as *dilarangkan*. We may opt for a compulsory adverbial like *already* to make a past tense explicit.

4.3 Implementation

Our CNL is implemented in Grammatical Framework (GF, Ranta, 2004), a programming language for multilingual grammar applications. A grammar specification in GF consists of an abstract syntax, with one or more concrete syntaxes that contain linearization rules for the ASTs. The mapping is bidirectional: a GF AST is linearized to various strings admitted by different concrete syntaxes, and a string is parsed into all the ASTs that generate it.

Inspired by Ranta (2014), the base of our CNL is the GF Resource Grammar Library (RGL, Ranta et al., 2009), a wide-coverage syntactic library for over 30 languages. We presented modifications that make the CNL unambiguous in Section 4.2. In addition to those, the CNL contains other custom extensions, some of which are explained below.

- Accept predicates with placeholders: `{Business}` *provides* `{Service}` *;* `{Business}`*'s jurisdiction is* `{Country}` .
- Accept predicates without arguments: *held as representative of* is treated as `{X}` *is held as representative of* `{Y}`.
- Accept predicates that require a copula without one: *prohibited*, *is prohibited* and `{X}` *is prohibited* produce identical results.
- Valency is determined by the predicate's arity, as explained in Section 4.2. In a GF lexicon, each word has its own valency, so `V` and `V2` are different categories and can't be used interchangeably; the same applies for the pairs `N`–`N2` and `A`–`A2`. In our CNL grammar, we relax the subcategory rules when parsing: this leads to many redundant parses, but we filter them out based on the predicate's arity.
- Ad hoc constructions on top of the grammar to accommodate for legal language.

The CNL has two concrete syntaxes for English: one with the features listed in Section 4.2, and one without. Thus, if we parse *business or trade in Bali* in the imprecise concrete syntax, we get two trees, which are linearized unambiguously in the precise concrete syntax: *[(business or trade) in Bali]* and *(business or [trade in Bali])*. We will use this to our advantage: let the user type imprecise description, and then transform it into a precise CNL version
  ```

1. *Sharing* is noun:

   1a [involves with u.p.] sharing-fees
   1b involves [sharing-fees with u.p.]

2. *Sharing* is verb:

   2a [involves with u.p.] sharing fees
   2b involves [sharing with u.p.] fees
   2c involves sharing [fees with u.p.]

Figure 4: All different parses for *involves sharing fees with unauthorized persons* in precise CNL

for interactive disambiguation, similarly to methods used in discriminant-based treebanking (Carter, 1997), but aimed at non-linguists.

5 Parsing user input

Let us return to Figure 2, with the L4 predicate `UnauthorizedSharingFees`. As a first step, we parse the description *"involves sharing fees with unauthorized persons"*. For now, we assume that it parses—robust fall-back is future work.

The description is ambiguous in two orthogonal ways: part-of-speech for *sharing* and attachment of the adverbial. These two ambiguities result in five different parses, shown in Figure 4. We disambiguate the description interactively, and verify the answer by rendering the result in the precise CNL.

Top-level ambiguity The first step in disambiguation is to temporarily remove all postmodifier adjuncts (see Table 1) from the initial trees, thereby reducing the tree depth until the heads with light modifiers remain. The goal is to sieve out the main structure, and make it easier for the user to disambiguate one feature at a time.

In our example *"involves sharing fees with unauthorized persons"*, removing the postmodifier adjunct *"with unauthorized persons"* reduces the five trees to two: compound noun (*sharing-fees*) vs. verbal complement (*involves sharing*). We transform the trees into disambiguation alternatives and show the user:

1. the business involves sharing-fees
2. the business shares fees

To get these disambiguation alternatives, we have hand-crafted a large number of very specific transformation functions, which manipulate the

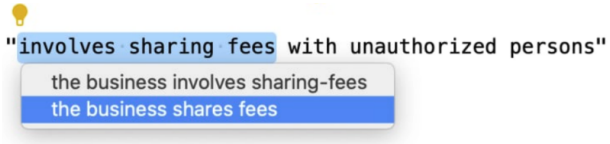


Figure 5: Future editor support (mockup)

sentence directly at the GF abstract syntax tree. For example, the following transformation applies to any sentence with the same structure.

- Match a finite transitive verb (*involves*) with a gerund complement (*sharing fees*).
- Remove the original finite verb, and transform the gerund verb phrase into finite verb phrase (*shares fees*).

In addition, we apply a subject to all of the alternatives. As seen in the examples, we use the noun *business*, which we get from the type signature of the predicate, `Business → Bool`.

Disambiguate adjuncts internally After the top-level disambiguation, we check whether the temporarily excluded adjuncts have something to disambiguate internally.

Suppose, for the sake of example, that the original predicate was, instead *involves sharing fees with any business or trade in Bali*. In such a case, we first disambiguate the internal structure of the adverbial (*with any business or...*), using the same method as before: apply another set of transformation functions to the adjuncts to produce disambiguation alternatives, and ask the user which one they meant.

Sometimes, the difference in adjuncts is linked to the difference in the main tree, in which case, we're done after picking the correct adjunct. But in our example case, the ambiguity is orthogonal to the adverbial attachment, so we need one more step.

Disambiguate adjunct attachment Once we have disambigated everything else—in the main tree or recursively in the subtrees—the remaining ambiguities (if any) should be related to attachment. If the user chose *sharing* to be a verb, we present the following alternatives.

a) involves with unauthorized persons
b) shares with unauthorized persons
c) fees with unauthorized persons

an old astronomer ~~with a telescope~~	Adverbial *with a telescope* removed. Determiner and adjective are kept, because they are premodifiers.
sleeps ~~on a soft mattress~~	*on* introduces an adjunct. The whole adjunct is removed.
depends on legal work ~~performed by a legal practitioner~~	*on* introduces an obligatory complement. The complement *on legal work* is kept, but internally reduced.

Table 1: Examples of postmodifier adjuncts that are temporarily removed for the first stage of disambiguation

Verifying the answer After the interactive disamgibuation, we show the result: *involves [sharing with unauthorized persons] fees*. If the user is unhappy with the result, they may revert the disambiguation and start over.

Once a user is familiar with the precise CNL, they may write the initial description with it, and hence skip the interactive process. The IDE tries to parse the input as both variants.

6 Future work

At present, we have worked on the CNL only inside the project, occasionally consulting an expert legal knowledge engineer, who is a part of our team. Next steps are to find a real-life use case with a more diverse team of users and conduct an evaluation. We expect that we need to change the CNL, as it goes from our hands to the hands of non-experts. It will be interesting to see whether it also needs to change from domain to domain: aside from different lexicon, would two unrelated fields, such as insurance and construction site regulations, need any changes in the core CNL?

In order to make L4 and the CNL easier to use, we will add editor support and robust fall-back options. We will also create a specification and resources for user guidance for the CNL, and add languages other than English.

IDE support L4 has a plugin for Visual Studio Code, and we plan to integrate editor support for the CNL as well. Figure 5 shows a sketch of a potential user interaction. We may also experiment with hovering over single words to provide information about part-of-speech, or show dependency structure—for instance, an arrow from *with unauthorized persons* to all of its possible heads.

Robust fall-back options First, suppose that the natural language description is parsed in the host grammar, the GF RGL, but there are no transformation functions that spell out the ambiguities (as explained in Section 5), nor is it covered by the precise CNL. We will experiment with alternative methods of disambiguation, such as choosing a part of speech for individual words and showing dependency relations between words. These require more knowledge about grammar, but they work for any tree that is successfully parsed, and are thus more robust.

Currently lexicon entries are generated from a WordNet lexicon in GF (Angelov, 2020), and a few words that we have added after finding them in the sources of our small experiments. Where a word or phrase cannot be parsed from a pre-existing lexicon, the user will be prompted to define every out-of-vocabulary word, and include it in a user-defined lexicon. We will use GF's smart paradigms (Détrez and Ranta, 2012) to make an initial guess from a limited number of word forms, with a possibility to correct any wrong guesses.

In the case that parsing still fails, the sentence is syntactically out of scope from the CNL. Already now, the parser could try to break the sentence down into phrases that can be parsed, but we would need to guide the user how to reconstruct the whole sentence to be within the scope. We recognize the limits of the GF parser in handling ungrammatical output, and may look into more robust parsing pipelines, such as dependency trees to GF abstract syntax (Kolachina and Ranta, 2016).

Multilingual support So far, we support the translation of L4 to English, with a view to support multiple languages in the near future. With multilingual output, we need to also support word sense disambiguation and multi-word expressions. For instance, a term like *sole proprietor* will be wrong in many languages, if translated compositionally: "lonely proprietor" instead of the correct multi-word term. As we work on a use case in a specific domain, it will be more realistic to have a good coverage for a lexicon.

As for word sense disambiguation, our short-term plan is to use WordNet (Fellbaum, 1998) glosses to ask the correct sense for each word. Longer-term, we may look into statistical methods for guessing the most likely word senses.

Acknowledgements

This research is supported by the National Research Foundation (NRF), Singapore, under its Industry Alignment Fund — Pre-Positioning Programme, as the Research Programme in Computational Law. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Krasimir Angelov. 2020. A Parallel WordNet for English, Swedish and Bulgarian. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 3008–3015.

Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. 2018. Constraint answer set programming without grounding. *Theory and Practice of Logic Programming*, 18(3-4):337–354.

David Carter. 1997. The TreeBanker: A tool for supervised training of parsed corpora. In *Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, pages 9–15, Madrid, Spain.

Marina De Vos, Doğa Gizem Kisa, Johannes Oetsch, Jörg Pührer, and Hans Tompits. 2012. Annotating answer-set programs in LANA. *Theory and Practice of Logic Programming*, 12(4-5):619–637.

Grégoire Détrez and Aarne Ranta. 2012. Smart paradigms and the predictability and complexity of inflectional morphology. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 645–653, Avignon, France. Association for Computational Linguistics.

Docassemble. `https:/docassemble.org` [online].

Chris Dornan and Isaac Jones. Alex user guide [online]. 2003.

Christiane Fellbaum. 1998. *WordNet: An electronic lexical database*. MIT press.

Norbert E. Fuchs and Rolf Schwitter. 1996. Attempto Controlled English (ACE). In *Proceedings of the First International Workshop on Controlled Language Applications*.

Prasanth Kolachina and Aarnte Ranta. 2016. From abstract syntax to universal dependencies. In *Linguistic Issues in Language Technology, Volume 13, 2016*.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational linguistics*, 40(1):121–170.

Inari Listenmaa, Jason Morris, Alfred Ang, Maryam Hanafiah, and Regina Cheong. 2021. An NLG pipeline for a legal expert system: a work in progress.

Nathaniel Love and Michael Genesereth. 2005. Computational Law. In *Proceedings of the 10th international conference on Artificial intelligence and law*, ICAIL '05, pages 205–209, New York, NY, USA. Association for Computing Machinery.

Simon Marlow and Andy Gill. Happy user guide [online]. 1997.

Denis Merigoux, Raphaël Monat, and Jonathan Protzenko. 2021. A Modern Compiler for the French Tax Code. In *CC '21: 30th ACM SIGPLAN International Conference on Compiler Construction*, pages 71–82, Virtual, South Korea. ACM.

Jason Morris. 2021. Constraint Answer Set Programming as a Tool to Improve Legislative Drafting: A Rules as Code Experiment. In *ICAIL '21: Proceedings of the 18th edition of the International Conference on Articial Intelligence and Law*.

OpenFisca. `https://openfisca.org` [online].

Aarne Ranta. 2004. Grammatical Framework. *Journal of Functional Programming*, 14(2):145–189. Publisher: Cambridge University Press.

Aarne Ranta. 2011. Translating between language and logic: what is easy and what is difficult. In *International Conference on Automated Deduction*, pages 5–25. Springer.

Aarne Ranta. 2014. Embedded controlled languages. In *International Workshop on Controlled Natural Language*, pages 1–7. Springer.

Aarne Ranta, Ali El Dada, and Janna Khegai. 2009. The GF Resource Grammar Library. *Linguistic Issues in Language Technology*, 2(2):1–63.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.

Rolf Schwitter. 2012. Answer set programming via controlled natural language processing. In *International Workshop on Controlled Natural Language*, pages 26–43. Springer.

M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. 1986. The British Nationality Act as a logic program. *Communications of the ACM*, 29(5):370–386.

Doug Skuce. 2003. A controlled language for knowledge formulation on the semantic web.

Assessing and Enhancing Bottom-up CNL Design for Competency Questions for Ontologies

Mary-Jane Antia
Department of Computer Science
University of Cape Town
South Africa
mantia@cs.uct.ac.za

C. Maria Keet
Department of Computer Science
University of Cape Town
South Africa
mkeet@cs.uct.ac.za

Abstract

Competency questions (CQs) are used in ontology development to demarcate the scope, provide insights into their content, and verification. Their use has been impeded by problems with authoring good CQs. This may be assisted by a controlled natural language (CNL), but its development is time-consuming when carried out manually. A recent study on data-driven CNL design to learn templates from a set of CQs, resulting in CLaRO, had somewhat better coverage and some noise due to grammar errors in the source CQs. In this paper, we aim to investigate such a bottom-up approach to CNL development for CQs regarding the effects of 1) improving the quality of the source data 2) whether more CQs from other domains induce more templates and 3) if the structure of knowledge in subject domains has a role to play in the matching of patterns to templates; therewith might indicate that possibly a structure of knowledge in a subject domain may continue to affect bottom-up CNL creation. The CQ cleaning increased the number of templates from 93 to 120 main templates and an additional 12 variants. The new CQ dataset of 92 CQs generated 27 new templates and 7 more variants. Thus, increasing the domain coverage had the most effect on the CNL. The *CLaRO v2* with all generated templates has 147 templates and 59 variants thereof and showed 94.1% coverage.

1 Introduction

Competency questions (CQs) are natural language expressions which are used among others in the design, development and verification of ontologies (Suárez-Figueroa et al., 2012). CQs have attracted research interest in ontology engineering since the mid 1990s and are noted as a key requirement for ontology development by methodologies such as the NeON methodology (Suárez-Figueroa et al., 2012). They have been shown to serve different purposes in the engineering of ontologies, including demarcating their scope, providing insights into their contents, and ensuring their answerability (Ren et al., 2014). However, there is limited acceptance and use of CQs. Several reasons have been proposed for that. There is limited support for authoring CQs (Keet et al., 2019; Ren et al., 2014) and their translatability into queries over the ontology or candidate axioms. Also, CQs developed for one ontology or subject domain often cannot be reused for another related ontology or domain, which impedes the re-usability of CQs (Fernández-López et al., 2019). This spurred the inquiry into controlled natural language (CNL)-assisted CQ authoring as a holistic solution to these issues, including archetypes (Ren et al., 2014), patterns (Bezerra et al., 2014), and the template-based Competency question Language for specifying Requirements for an Ontology, model, or specification (CLaRO) (Keet et al., 2019) that has been shown to have broader coverage than the former two.

CLaRO was developed in a bottom-up method using a dataset of 234 CQs for five ontologies as-is together with NLP-based sentences analyses (Potoniec et al., 2020; Wiśniewski et al., 2019). However, as is well-

known for other data-driven tasks, the quality of the output is dependent on the coverage and quality of the data and the quality of the algorithms. In this case, manual inspection of the CQs indicated grammar issues, the CQs were for only five ontologies, and the sentence chunking was not closely investigated. Since methods for data-driven template creation from 'small data' for specialised tasks still can be useful to be able to do, we aim to investigate this in more detail. To this end, this paper seeks to answer the following research questions:

1. What is the effect of 'cleaning' (correcting) CQs to the set of templates in CLaRO?
2. What are the effects of increasing the number and diversity of CQs to the template development?
3. What role does the structure of knowledge in subject domains play in the matching of patterns to templates, if any; and if so, how?

Correcting CQs has been carried out manually, whereas for the second question, we collected 70 new CQs related to ontologies in different domains and added 22 newly formed CQs inspired by some CQs in the initial CQs set. They all went through the template design process. Unlike the manual only evaluation in the initial *CLaRO*, our evaluations have been automated with the same chunking algorithm, intended as a step toward increased automation of CQ authoring and use and automation of CNL evaluation.

CQ correction applied to 9% of the original dataset and 17.8% of the new CQ set (the 70 CQs for different ontologies), of which the effects on the number of templates are 10 and 8 templates, respectively. The new CQs for the *Cleaned CLaRO* had a coverage of 40% and upon adding those 52 new templates and 19 variants generated from them, reaching 147 templates and 59 variants in total, its coverage reached 94%, compared to 88% for just ClaRO. Increasing domain coverage further thus had a bigger impact on CNL design than better source data.

The remaining sections of the paper are structured as follows: related work is described in Section 2; the methodology in Section 3; results and discussion in Sections 4; and conclusions in Section 5.

2 Related Work

The importance of CQs in ontology engineering has been documented with a focus on their use as part of the requirement specification in ontology development (Bezerra and Freitas, 2017; Keet and Lawrynowicz, 2016; Suárez-Figueroa et al., 2012), as noted in Section 1. There is limited evidence of uptake of CQs, however, and a substantial number of CQs have a range of issues (Potoniec et al., 2020), such as being unanswerable by an ontology and grammar. Since CQs are typically developed for specific ontologies and no guidelines exist for authoring, this continues to affect the quality of CQs and hamper the uptake by ontology engineers.

To address the lack of authoring support, a set of 12 core and 7 variant archetypes from 150 CQs for the Pizza and SWO ontologies has been proposed (Ren et al., 2014). The variables in their "archetypes" (templates) are ontology elements (OWL class and object property), rather nouns and verbs, therewith narrowing the use to OWL and having a 1:1 mapping to the ontology that dictates axiomatisation. Bezera et al. identified 14 patterns and 3 CQs types, also for use in only OWL ontologies ontology elements for variables. Their set omits the 'Who, Where' question types in their identification, even though these question types exist within their source data CQs used (also for the Pizza ontology), further limiting the coverage (Bezerra et al., 2014).

Other studies include (Malheiros et al., 2013), who use of grammatical tags with a set of rules for the CQs, which is limited to three predefined types (is-a, yes/no, and existence question) and also contain a 1:1 mappings to the ontology. Wisniewski et al. created 106 patterns through a process where the linguistic structures of 234 CQs from 5 different ontologies (Dem@care, Stuff, AWO, OntoDT and

SWO) were chunked using NLP methods, replacing the nouns and verbs with the terms entity chunks and predicate chunks (Wiśniewski et al., 2019; Potoniec et al., 2020). These 106 patterns were used to develop the initial CLaRO CNL templates (Keet et al., 2019). The patterns from (Wiśniewski et al., 2019; Potoniec et al., 2020) and CLaRO and templates do not have 1:1 mappings or CQs type restriction and are thus not only for OWL ontologies. With more CQs, it showed to have better coverage (Keet et al., 2019), suggesting that an even larger set of CQs may increase coverage further.

3 Methodology

To begin this section, we provide an overview of how the initial *CLaRO* was created which also forms part of the method in this study. Note that the CQs used in *CLaRO* and for this study were only in the English Language.

3.1 Preliminaries: CLaRO development

The *CLaRO* templates of (Keet et al., 2019) were developed using patterns obtained from 234 CQs drawn from 5 different ontologies through a semi-automated process in a previous study conducted by (Wiśniewski et al., 2019; Potoniec et al., 2020). The authors of the patterns employed a linguistic approach in understanding the structure of CQs in order to create an abstract identification of each CQs since most of them were different from one another in their natural language form. Two main text chunks were used to represent the linguistic structures. They were called entity chunk (for a noun or noun phrase), and predicate chunk (which represents a verb phrase). By doing so, CQs with identical structures were grouped together and considered as patterns. The patterns used were identified and implemented in the following manner:

1. CQs were manually checked to ensure that they were T-BOX questions.
2. CQs were divided into two types: materialised (were the entities are embedded in the CQs) and dematerialised (were the entities are replaced by a placeholder such as $task\ X$, it, $datatype\ Y$, $gene\ X$.
3. After CQs are chunked into patterns, if the same structures occurs in more than one CQs, be it within or across the ontologies, they are selected as patterns. Individual CQs chunks are referred to as "candidate patterns".
4. Text chunks from dematerialised CQs (i.e., CQs with placeholders) are considered as patterns even if they are observed only once.
5. Text chunks from materialised CQs (i.e., CQs without placeholders) can only be considered as patterns if they are observed more than once in the entire CQs.

This resulted in 106 patterns (Wiśniewski et al., 2019), which were used in the development of the templates of *CLaRO*. The design for that CLaRO also included tackling issues such as redundant words and pronouns in templates, generating additional templates to cater for negation, handling of plural/singular forms as well as synonym usage (Keet et al., 2019). There were 93 templates and 40 variants in *CLaRO* before this study was conducted.

3.2 Current Design and Evaluation Process

Following the same semi-automated, data-driven bottom-up approach on which the initial *CLaRO* (referred to as *CLaRO v1* from now on) was designed (Keet et al., 2019), a series of activities were designed to assess grammar quality of base CQs dataset for *CLaRO v1* and the effect of this, possible increase in the number of templates created as a result of increases number and diversity of CQs. These activities are presented as three steps carried out in the design. Fig. 1 gives a pictorial representation of the processes described below.

3.2.1 Step1: Cleaning and Verification

We begin by assessing the CQs dataset used in pattern development stage for the CLaRO v1 templates (Potoniec et al., 2020; Keet et al., 2019). The first task thus was centered around cleaning the CQs set. The original CQs dataset was sent to a linguist for analysis, who re-

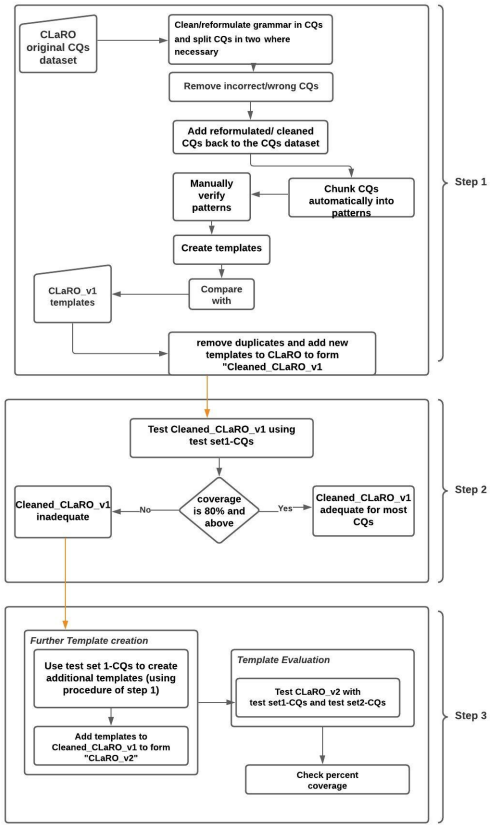


Figure 1: Cleaned CLaRO and CLaRO v2 development procedure

viewed them and provided suggestions for corrections. This grammar analysis step of the cleaning process was carried out manually. The incorrect questions were removed from the dataset while the corrected form of the CQs and any new CQs which arose from question splits were added to the dataset (e.g., due to splitting up a long sentence). Subsequently, this new CQs dataset were then automatically chunked and verified automatically with the algorithm used. Afterwards, a manual verification was done by checking the candidate patterns generated against the CQs from which they were generated. For 5 patterns, manual corrections were made; for example: *What PC1 EC1 PC2 EC1 PC2?* had the original chunking failing to pick up 'algorithms' as an *EC* (*What PC1 algorithms PC2 EC1 PC2?*). Another example is: *How well [PC1] is [EC1] for [EC2]?* where the original chunk failed to pick up *documented* as a *PC* (*How well documented is [EC1] for [EC2]?*). With cleaning and verification of patterns completed, new templates were developed and then compared with the existing *CLaRO v1* templates. Similar occurrences were also observed with the newly sourced Evaluation CQs discussed in step2. Templates found in the new set that were also found in *CLaRO v1* were removed, the remaining were added to the *CLaRO v1* templates to make up what called the *Cleaned CLaRO v1* templates.

3.2.2 Step2: Template Evaluation

The next step was to test the *Cleaned CLaRO v1* using a new set of CQs. To select CQs, we searched widely to find ontologies with accompanying research papers that listed their CQs. Some of them were found within the papers while others pointed to Github repositories where the CQs could be found. It is unknown if the CQs were carefully reviewed. They did have some grammar-related corrections and most of them were CQs that could be answered by an ontology were publicly available. These CQs were collected as our set of new CQs. All of the new CQs were not part of any previous CQs used in *CLaRO v1*. The larger portion of the CQs were set aside for the first test and labeled as *test-set1*. These CQs come from 4 ontologies which include xAPI ontology (CQs=6) (Vidal et al., 2018), MEMON ontology (CQs=2) (Masmoudi et al., 2018), EM-KPI ontology (CQs=10) (Li et al., 2019) and INHD ontology (CQs=52) (Stucky et al., 2019). Another set of 22 new CQs inspired by a part of the initial CQs dataset in the *CLaRO v1* CQs processing phase were authored in line with recommendations form the linguist. These 22 additional CQs were added because they were semantically similar to the original CQs in the dataset and they serve to address CQs authoring preferences i.e., bring flexibility to authoring styles of asking questions which produce the same answers in the ontology. They were added to the *test-set1* CQs bringing the total to 92 CQs. These CQs were chunked to obtain patterns (note:

such chunked sentences are called 'candidate patterns' in (Wiśniewski et al., 2019)), which were manually verified, and then automatically checked against the set of templates to see if they matched any of the *Cleaned CLaRO* templates. For this test purpose, we set an adequacy benchmark of 80%. If it is lower, then the *test-set1* CQs should be used as a dataset to create additional templates.

3.2.3 Step3: Final Curation

This step has two processes: further template creation and template evaluation. If the evaluation carried out in step2 shows the coverage to be adequate (i.e., up to or more than the benchmark set), then the step3 template evaluation process is carried out immediately after. However, if the it shows the coverage to be less than the desired benchmark, the additional template creation process will take place before the template evaluation process in done. On carrying out the step2 evaluation (see results below), the coverage was less than required and thus, further template enhancement was needed. We repeat the procedure in step1, however, this time we make use of the *test-set1* CQs as our dataset. The template comparison was done between the newly created templates and *Cleaned CLaRO v1* templates. The combination of any new templates found in this process and the *Cleaned CLaRO v1* templates form what we shall call *CLaRO v2* templates. Finally, the step3 template evaluation process is carried out and then we ascertain the coverage in relation to our benchmark. It is worth nothing that the domain knowledge areas of the ontologies used for creation of *CLaRO v2* ranges from software, stuff, to dementia as part of the initial set of CQs in *CLaRO v1* to knowledge domains of the ontologies for the *test-set1* CQs that included energy management, environmental analytics, insects natural history, object oriented code, depression care and biomedical. To evaluate *CLaRO v2*, a second test was carried out using a combination of *test-set1* CQs and *test-set2* CQs (which consist of the remaining part of the newly sourced CQs) obtained from (Jung et al., 2017; Azzi et al., 2019; de Aguiar et al., 2019) ontologies. Using the same benchmark and partially automated test process as in step 2, we check for the percentage coverage of our results.

The data, code, and results are available from `https://github.com/mkeet/CLaRO`.

4 Results and Discussion

To start with step1 of the method section, 22 out of the 234 CQs were identified with either grammar related issues or could not be answered in an ontology in its current state and so were reformulated from the 234 CQs set used in creating the patterns in the *CLaRO v1* study; see samples in Table 1. Another example of a problematic question (an open-ended question) encountered was: *To what extent does [the software] support appropriate open standards?* which is reformulated as: *Does [the software] support open standards?*. These CQs as stated in the method section, were reformulated or split were necessary and added to the dataset; their original formats were removed from the dataset.

The number of patterns identified were 145 (which gives an additional 39 patterns to the previous 106 patterns in the previous study). As with the original study, after some manual cleaning of the patterns were necessary, all the patterns found that met the design decisions were included as templates, no new negation templates were added. Having obtained the patterns and proceeded to create new templates, on comparing the new templates with *CLaRO v1*, we found that most of the resulting templates were present. When the templates are put together, an added 27 templates and 12 variant to bring the total to 120 templates and 52 variants, now referred to as *Cleaned CLaRO v1* templates.

In Step2, a new set of CQs were used to carry out a first test on the *Cleaned CLaRO v1* templates. After chunking and cleaning, their patterns were compared with the templates in a bid to find a match. Of the 92 CQs in *test-set1*, 38 patterns were found not to present in *Cleaned CLaRO v1*. Given our benchmark percentage set at the beginning of the study, a

40% coverage was inadequate to declare that *Cleaned CLaRO v1* was sufficient for most unseen CQs. Thus, the *test-set1* CQs were used as a dataset for the creation of another set of templates. Following the procedure outlines in the preliminary section above and applied in step1 of the method section, 35 patterns were found to have fulfilled the design decisions and were included as templates. When the new templates were compared with *Cleaned CLaRO v1*, four were found to be present.The rest 34 templates when split into actual templates and variants, are 27 templates and 7 variants. We then combined them to *Cleaned CLaRO v1*, making the total number of 147 templates and 59 variants. This new total of templates were now called *CLaRO v2*.

For the evaluation of *CLaRO v2*, *test-set2* CQs ($n = 26$) from (Jung et al., 2017; Azzi et al., 2019; de Aguiar et al., 2019) ontologies as well as *test-set1* CQs were used. CQs that were removed include *How are classes logically organized in an OO source code?*, since no ontology will be able to answer this due to its descriptive nature. One duplicate question was removed: *What are the signs and symptoms of adolescent depression?* and *What are the physical symptoms of adolescent depression?*. The overall results show that 111 of the 118 CQs had their patterns present in the *CLaRO v2* templates on first try, i.e., 94.1%, with a few of the matching templates coming from *CLaRO v1* and the rest templates found in the greater *CLaRO v2*. With this result surpassing our 80 percent benchmark set in step 2 of the method section, we accept *CLaRO v2* as being adequate for most unseen CQs. Table 2 shows the results from the first test on *Cleaned CLaRO v1* as well as the results of the second test on *CLaRO v2*.

4.1 Discussion

We attempt to answer the first research question which addresses the role of cleaning in the identification of templates, we saw from the results that having reformulated and splitting problematic CQs, some of their patterns resulted into templates in *Cleaned CLaRO v1*.

Table 1: Sampling of reformulated CQs.

Grammaticality
1. What are the values of a rain properties (unit, location, date, etc.)? **Comment:** *Incorrect English, illustration* **Reformulated CQ and pattern:** Where are the property values of a rainfall? *What are EC1 of EC2?* (new template: Yes)
2. Is [it] open source or not? **Comment:** *Informal writing* **Reformulated CQ:** Is [it] open source? *Is EC1 EC2?* (new template: No)
Answerability
1. How do I get help with [it]? **Comment:** *A descriptive question* **Reformulated CQs and pattern:** What are the sources of support for [it]? *What are EC1 of EC2 for EC3?* (new template: No)
2. How can I get problems with [it] fixed? **Comment:** *A descriptive question* **Reformulated CQ and pattern:** Who can fix problems with [it]? *Who PC1 EC1 with EC2?* (new template: No)
Two-in-one question
1. Do I know anyone who has used [this software] or processed [this type of data]? **Comment:** *A personalized question (also two questions in one)* **Reformulated CQs and pattern:** a. Who has used [this software] in the past successfully? *Who PC1 PC1 EC1?* (new template: Yes) b. Who has processed [this type of data]? *Who PC1 PC1 EC1 in EC2 EC3?* (new template: Yes)

The second research question which aims at assessing if there are identifiable linguistic patterns to knowledge structures in different subject domains and if these patterns influenced the matching of CQs patterns to CLaRO v2 templates. We looked at our results in terms of their distribution according to the domains of the ontologies in the *CLaRO v2*, this is to determine how much of the test CQs patterns were matched within the templates that made up *Cleaned CLaRO v1*. The results showed that most of the patterns where matched in *CLaRO v2*, leaving only a few in *Cleaned CLaRO v1* alone. For the third research question which aims at assessing how the increased number and diversity of CQs from more ontologies

Table 2: Evaluation results of Cleaned CLaRO v1 versus CLaRO v2 using generated patterns from test CQs

Cleaned CLaRO v1	Count	Coverage%
Absent	57	60%
Present	38	40%
CLaRO v2	Count	Coverage%
Absent	7	5.9%
Present	111	94.1%

impacts the template creation. A total of additional of 54 templates and 19 variants were created and added to the *CLaRO v1*'s 93 templates and 40 variants to form the 147 templates and. Consequently, a wider coverage was achieved for CQs from many different ontologies as seen in the test results.

Also worth reporting in our findings is the distribution of templates in terms of knowledge domains; we analysed *CLaRO v2* results using the test-set2 CQs which only came into use as part of the second test in our study, we found that majority of the templates matched in this group were found to be specifically in *Cleaned CLaRO v1* templates. On further observation, we also found the templates matched patterns from the SWO and Dem@care CQs. This observation may be linked to the fact that the Knowledge domains of the test-set 2 CQs were from adolescent depression, object oriented code and biomedical (which had very few CQs compared to the other two), and these domains are some-what related in knowledge to the knowledge found software and Dementia. For instance *What are [EC1] for [EC2]?* from SWO number 19, *What [PC1] [EC1]?* from Dem@care number 146 and *What [EC1] [PC1] [EC2]?* from SWO number 12 patterns can be observed in the test-set2 patterns. With the test-set2 CQs alone, *CLaRO v2* templates all patterns from the depression care CQs were present; object oriented code CQs and the biomedical knowledge domains also most of their CQs patterns present. The results can be seen in Table 3.

There were some situations where the limitation of the algorithm produced some strange pattern outputs like *Who else [PC1] [PC1] [EC1] [PC1]?* for *Who else has used [tool x] today?* which would clearly give a different pattern when it is generated manually. Another example is the presence of past tense in CQs which not handled properly, as seen with *Has [species X] been collected at lights?* which produces *[PC1] [EC1] been [PC1] [EC2]?*. With the breakdown of the final results in this study showing over 90% coverage for *CLaRO v2* and the diversity of the domains which the CQs/ontologies were drawn from, we assume that although the knowledge structure in domains may play a role in the matching of templates to CQs patterns, that role is minimal in our study.

Ontologies are known to represent real world complexities using the knowledge structures they contain (Litovkin et al., 2018; Hnatkowska et al., 2020). The use of CQs as a functional requirement for ontology development makes it possible for ontologies to capture holistically, the knowledge in the given domain or sub domain. With the very general nature of the *CLaRO v2* templates, it is expected that they serve as a guide for ontology developers as they author their own CQs developed across different domains. Given the coverage obtained in this study, there may be an increase of the willingness to make use of CQs, and also a reduction in the frustrations that have characterised developers' experiences in authoring good quality CQs. Also, these templates potentially move us closer to achieving re-usability of CQs.

The results of this study will also enable ontology developers construct CQs that can provide users adequate information on the content of the ontology and at the same time provide the ontology with a quality verification tool. This study also shows the feasibility of bottom-up approaches to CNL design with better insight derived from these methods. *CLaRO v2* will enable subject experts and ontology developers develop CQs that are suitable for ontologies to answer and possibly give ideas of

Table 3: Results of Cleaned CLaRO v1 compared with CLaRO v2 on matched test-set2's chunked CQs representation, separated by domain.

CLaRO	Depression	Biomedical	OO code
Cleaned CLaRO v1	7 of 9 (78%)	2 of 7 (28.6%)	4 of 6 (66.7%)
CLaRO v2	9 of 9 (100%)	5 of 7 (71.4%)	5 of 6 (83.3%)
Absent	0	2 of 7 (28.6%)	1 of 6 (16.7%)

what other forms of CQs could still be derived for the ontology that may not have been considered. Although not the focus of this study, domain knowledge structures have become important to the advancement of CNL for CQs. With *CLaRO v2* now containing several new templates which has base CQs drawn from a range ontologies from different knowledge domains, we can expect that CQs patterns from more knowledge domains will be sufficiently catered for with the templates in *CLaRO v2* increasing the possibility of that it indeed has the potential of representing templates across many different domains.

5 Conclusion

In cleaning the source data and extending the *CLaRO v1* templates, we have been able to show that there is sufficient reason to assume that CQ templates can be reused across ontologies. Increasing domain coverage of the source CQs has a larger effect on the quality and number of templates than correcting erroneous CQs. The extended *CLaRO v2* templates created on the basis of the CQs patterns derived from a total of 329 CQs for SWO AWO, Stuff, Dem@care, OntoDT, xAPI, EM-KPI, inhd and memon ontologies, and has recorded a 94.1% accuracy.

The templates can assist CQ authors in writing good CQ that should be answerable by an ontology. Future work includes investigating further the role that domain knowledge structures play in the matching of CQs patterns and the possibility of identifying more templates. To achieve this would mean being intentional on seeking to identify the representation of CQs from a wide range of knowledge domains that are clearly unrelated to the those present in *CLaRO v2*. We also plan to update the CLaRO CQ editor tool with the new templates.

Acknowledgments

This work was financially supported by Hasso Plattner Institute for Digital Engineering through the HPI Research School at UCT. Many thanks to Prof BE Antia for his help with the linguistic analysis.

References

Camila Zacché de Aguiar, Ricardo de Almeida Falbo, and Vítor E Silva Souza. 2019. Ooc-o: A reference ontology on object-oriented code. In *International Conference on Conceptual Modeling*, pages 13–27. Springer.

Sabrina Azzi, Michal Iglewski, and Véronique Nabelsi. 2019. Competency questions for biomedical ontology reuse. *Procedia Computer Science*, 160:362–368.

Camila Bezerra and Fred Freitas. 2017. Verifying description logic ontologies based on competency questions and unit testing. In *ONTOBRAS*, pages 159–164.

Camila Bezerra, Filipe Santana, and Fred Freitas. 2014. Cqchecker: a tool to check ontologies in owl-dl using competency questions written in controlled natural language. *Learning & Nonlinear Models*, 12(2):4.

Mariano Fernández-López, María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. 2019. Why are ontologies not reused across the same domain? *Journal of Web Semantics*, 57:100492.

Bogumiła Hnatkowska, Adrianna Kozierkiewicz, and Marcin Pietranik. 2020. Semi-automatic definition of attribute semantics for the purpose of ontology integration. *IEEE Access*, 8:107272–107284.

Hyesil Jung, Hyeoun-Ae Park, and Tae-Min Song. 2017. Ontology-based approach to social data sentiment analysis: detection of adolescent depression signals. *Journal of medical internet research*, 19(7):e259.

C. M. Keet and A. Lawrynowicz. 2016. Test-driven development of ontologies. In *Proceedings of the 13th Extended Semantic Web Conference (ESWC'16)*, volume 9678 of *LNCS*, pages 642–657, Berlin. Springer. 29 May - 2 June, 2016, Crete, Greece.

C Maria Keet, Zola Mahlaza, and Mary-Jane Antia. 2019. Claro: a controlled language for authoring competency questions. In *Research Conference on Metadata and Semantics Research*, pages 3–15. Springer.

Yehong Li, Raúl García-Castro, Nandana Mihindukulasooriya, James O'Donnell, and Sergio Vega-Sánchez. 2019. Enhancing energy management at district and building levels via an em-kpi ontology. *Automation in Construction*, 99:152–167.

D. Litovkin, A. Anikin, M. Kultsova, and E. Sarkisova. 2018. Representation of what-knowledge structures as ontology design patterns. In *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–6.

Yuri Malheiros, Fred Freitas, and Rio Tinto-PB-Brazil. 2013. A method to develop description logic ontologies iteratively based on competency questions: an implementation. In *ONTOBRAS*, pages 142–153. Citeseer.

Maroua Masmoudi, Sana Ben Abdallah Ben Lamine, Hajer Baazaoui Zghal, Mohamed Hedi Karray, and Bernard Archimède. 2018. An ontology-based monitoring system for multi-source environmental observations. *Procedia Computer Science*, 126:1865–1874.

Jedrzej Potoniec, Dawid Wiśniewski, Agnieszka Ławrynowicz, and C Maria Keet. 2020. Dataset of ontology competency questions to sparql-owl queries translations. *Data in brief*, 29:105098.

Yuan Ren, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter, and Robert Stevens. 2014. Towards competency question-driven ontology authoring. In *Extended Semantic Web Conference (ESWC'14)*, LNCS. Springer.

Brian J Stucky, James P Balhoff, Narayani Barve, Vijay Barve, Laura Brenskelle, Matthew H Brush, Gregory A Dahlem, James DJ Gilbert, Akito Y Kawahara, Oliver Keller, et al. 2019. Developing a vocabulary and ontology for modeling insect natural history data: example data, use cases, and competency questions. *Biodiversity data journal*, 7.

Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. 2012. The neon methodology for ontology engineering. In *Ontology engineering in a networked world*, pages 9–34. Springer.

Juan C Vidal, Thomas Rabelo, Manuel Lama, and Ricardo Amorim. 2018. Ontology-based approach for the validation and conformance testing of xapi events. *Knowledge-Based Systems*, 155:22–34.

Dawid Wiśniewski, Jedrzej Potoniec, Agnieszka Ławrynowicz, and C Maria Keet. 2019. Analysis of ontology competency questions and their formalizations in sparql-owl. *Journal of Web Semantics*, 59:100534.

Human-understandable and Machine-processable Explanations for Sub-symbolic Predictions

Abdus Salam and **Rolf Schwitter** and **Mehmet A. Orgun**
Macquarie University, Sydney, Australia
{abdus.salam, rolf.schwitter, mehmet.orgun}@mq.edu.au

Abstract

HESIP is a hybrid machine learning system in which a sub-symbolic machine learning component makes a prediction for an image classification and afterwards a symbolic machine learning component learns probabilistic rules that are used to explain that prediction. In this paper, we present an extension to HESIP that generates human-understandable and machine-processable explanations in a controlled natural language for the learned probabilistic rules. In order to achieve this, the literals of the probabilistic rules are first reordered, and then aggregated and disambiguated according to linguistic principles so that the rules can be verbalised with a bi-directional grammar. A human-in-the-loop can modify incorrect explanations and the same bi-directional grammar can be used to process these explanations to improve the decision process of the machine.

1 Introduction

The recent success of machine learning (ML) models is remarkable, especially the success of sub-symbolic ML models in problem domains that are computationally expensive, such as natural language processing and image processing (Zhang et al., 2020; LeCun et al., 2015). Sub-symbolic ML models mostly learn functions that map the input data to the output data to find the correlations between them (Ilkou and Koutraki, 2020). When a ML model selects one or more class labels as an output for a given input, the output is known as a prediction of the model. These sub-symbolic models are used in intelligent systems to make better decisions based on their predictions. Since most of these sub-symbolic ML models are black-box models that are not immediately interpretable, it is difficult to explain to a user of an intelligent system how the machine learning algorithm came to a particular decision. This is why eXplainable AI (XAI) has recently gained momentum, since this discipline aims to produce explainable models that humans can understand, manage and trust (Gunning, 2017).

Most systems that can explain a prediction, such as Lime (Ribeiro et al., 2016) and Anchor (Ribeiro et al., 2018), explain the prediction based on features that exist in the dataset. For example, Lime selects super-pixels to explain an image prediction. This explanation may be helpful for a domain expert, but it may be difficult to understand for a non-expert user. Alternatively, researchers have tried to employ symbolic ML models to explain predictions, since these models are directly interpretable (Rabold et al., 2019). Symbolic ML models learn symbolic rules which are then presented to the user as an explanation for a prediction. Although these symbolic rules are easier to interpret as shown in several studies (Muggleton et al., 2018), they are sometimes difficult to understand by a user who does not have a background in formal methods. Therefore, it is important to study alternative ways of generating explanations that are human-understandable (and as we will argue later at the same time machine-processable).

In this paper, we present a linguistic extension to HESIP, a hybrid explanation system for image prediction. The extended implementation of the system uses a bi-directional grammar to generate explanations in a controlled natural language.

2 HESIP: System Description

HESIP combines sub-symbolic and symbolic representations in two separate components of the system to construct symbolic explanations for image predictions. According to Kautz's classification, HESIP is a hybrid system of Type-3 where a sub-symbolic component is used to work on a task, and then a symbolic component is used to finalise that task (Garcez and Lamb, 2020). HESIP is motivated by LIME-Aleph (Rabold et al., 2019) that is an explanation system where an image prediction is explained from the learned rules. The LIME-

Aleph system relies on two datasets that contain synthetic images. HESIP extends the architecture of the LIME-Aleph system and uses a more generalised approach so that it can be applied to real-world datasets. The architecture of the HESIP system is illustrated in Figure 1.

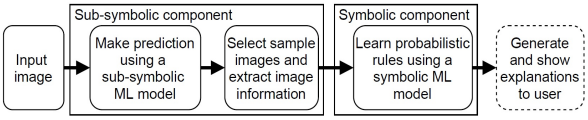


Figure 1: Architecture of HESIP

2.1 The Sub-symbolic Component

HESIP takes an image as input and makes a prediction with a probability using an artificial neural network (ANN) (Krizhevsky et al., 2012) as a sub-symbolic ML model. Based on the similarity to the input image, HESIP selects positive and negative instances of sample images. The probability of each of these sample images is predicted by the ANN. If the prediction probability of a sample image is greater than or equal to the prediction probability of the input image, then the sample image is a positive instance; otherwise, it is a negative instance. HESIP extracts the image information for all the positive and negative image instances. This image information is then used as observed data in the symbolic component of HESIP for learning probabilistic rules that explain a prediction.

Let us illustrate these steps in more detail using a motivating example from our own dataset that contains images of a particular shape. The task is then to determine if an image represents the concept of a house. Each image consists of two objects and each object has either the shape of a square or a triangle. The colour of these objects is either green or blue. We say that an image represents the concept of a house, if the image contains an object of type triangle that is located on the top of an (adjacent) object of type square. In any other case, the image does not represent the concept of a house (see Figure 2).

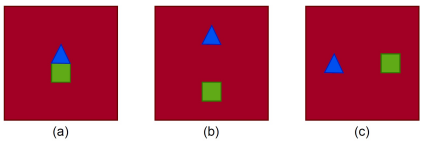


Figure 2: Examples of house concept learning. (a) represents the concept of a house, while (b) and (c) do not.

HESIP extracts the information from the image that will be used for the explanation; for example, information about the location of objects (e.g., position), about their properties (e.g., colour), and the relation between the objects, (e.g., on top of or left of). HESIP employs Detectron2 (Wu et al., 2019), a PyTorch-based object detection library that supports the Mask R-CNN method (He et al., 2017) to detect objects and their location information in an image. The location information is then used to extract the colour of the objects and to determine the relations between two objects in an image.

In our case, two objects can stand in the following relations: *left of*, *right of*, *top of*, *bottom of*, *on*, *under* and *contain*. Note that the relation *on* holds if an object is on the top of another object and these two objects are adjacent; similarly, the relation *under* holds if an object is beneath another object and these two objects are adjacent.

2.2 The Symbolic Component

HESIP employs a probabilistic logic programming framework called *cplint* (Riguzzi and Azzolini, 2020) as the symbolic component to learn probabilistic rules from data about positive and negative instances of sample images. The decision about whether an image is a positive or negative instance is provided by the sub-symbolic component. The probabilistic rules are then used to explain the prediction of an input image. These probabilistic rules have the following abstract form (Vennekens et al., 2004):

h_1`:`a_1 `;...;` h_n`:`a_n `:-` b_1`,...,` b_m`.`

where h_i are atoms, b_i are literals (incl. negation as failure), and a_i is the probability, a real number between `0` and `1`. The set of elements h_i`:`a_i forms the head of a rule and the set of elements b_i the body; the head and the body are separated by an *if*-symbol (`:-`). A disjunction in the head of a rule is represented with a semicolon (`;`) and atoms are separated by a colon (`:`) from probabilities.

Once the sample image information is available, HESIP represents the image using a simple ontology so that we can also generate explanations for other application domains using the same method. The ontology employed in HESIP consists of four predicates: (1) `object/1` represents an object; (2) `type/2` represents an object type; (3) `property/3` represents an object property; and (4) `relation/3` represents a relation between two objects. The probabilistic rules in HESIP contain only those predicates (atoms and literals) that are available in the ontology. In our case, the head of a rule may

contain the predicate `relation/3` or `type/2` and the body may contain any predicate of the ontology.

In our context, HESIP generates a probabilistic rule as shown in Listing 1 which states that an object A is of type house if all the conditions in the body of the rule are satisfied and the probability is `1`. Once a probabilistic rule is available, HESIP verbalises that rule in a controlled natural language (CNL) (Kuhn, 2014) and displays it together with the corresponding probability to explain the prediction of an input image.

Listing 1: A sample rule for house concept learning

```
type(A, house) : 1.0 :-
    type(B, triangle),
    object(B),
    type(C, square),
    object(C),
    relation(B, C, on),
    property(C, green, colour),
    property(B, blue, colour),
    relation(A, C, contain),
    relation(A, B, contain),
    object(A).
```

3 Generating Explanations

The rule in Listing 1 cannot be immediately verbalised, since the literals are not in an order that follows a linguistically motivated pattern. Our goal is to generate an explanation of the following form:

> *If an object contains a blue object of type triangle and contains a green object of type square and the blue object is located on the green object then the object is of type house.*

In order to achieve this, we use a bi-directional definite clause grammar similar to the one proposed by Schwitter (2018) that takes a set of reconstructed rules as input and generates explanations in a CNL as output. The same bi-directional grammar can be used to process a (modified) explanation and translate it into a rule as long as we stick to the syntax of the CNL. That means the CNL serves as a high-level interface language to the HESIP system and the user can modify and refine explanations and feed them back to the system.

3.1 Order of Content

Before we can verbalise the content of a rule, we need to identify those literals that introduce new content and distinguish them from literals that link to previously introduced content, and then reorder these literals according to a linguistic pattern. This process leads to a reconstruction of the rule where some literals are repeated so that they correspond to the underlying linguistic pattern. After reordering, the reconstructed rule looks as shown in Listing 2:

Listing 2: A sample rule after reconstruction

```
class(A, object), type(A, house) :-
    class(A, object),
    relation(A, B, contain),
    property(B, blue, colour),
    class(B, object),
    type(B, triangle),

    class(A, object),
    relation(A, C, contain),
    property(C, green, colour),
    class(C, object),
    type(C, square),

    property(B, blue, colour),
    class(B, object),
    type(B, triangle),
    relation(B, C, on),
    property(C, green, colour),
    class(C, object),
    type(C, square).
```

The body of this rule consists of three implicit linguistic patterns. The reordered literals in these patterns follow now a subject-verb-complement structure. The first two patterns use the same sequence of literal types and introduce new content; the subject position is occupied by a class, the verb position by a relation, and the complement position by a property, followed by a class and a type. The third pattern only uses new content in the verb position but previously introduced content in the subject and complement positions. The head of the rule consists of a pattern with a similar structure of the form subject-copula-complement where the subject position holds a class, the copula position is not filled, and the complement position holds a type. For this reconstructed rule, our grammar generates the following verbalisation:

> *If an object contains a blue object of type triangle and the object contains a green object of type square and the blue object of type triangle is located on the green object of type square then the object is of type house.*

This verbalisation is very explicit and can be improved using a number of micro-planning strategies (Reiter and Dale, 2000). However, since our goal is to generate explanations that are human-understandable as well as machine-processable, we

need to make sure that we do not introduce any ambiguities during micro-planning.

3.2 Aggregation of Content

Aggregation is the process of removing redundant information in a sentence (Dalianis and Hovy, 1993). In our case, we can use subject grouping to combine clauses and drop type information that has already been introduced to reduce redundancy (see Listing 3).

Listing 3: A sample rule after performing aggregation

```
class(A, object), type(A, house) :-
    class(A, object),
    relation(A, B, contain),
    property(B, blue, colour),
    class(B, object),
    type(B, triangle),
    relation(A, C, contain),
    property(C, green, colour),
    class(C, object),
    type(C, square),

    property(B, blue, colour),
    class(B, object),
    relation(B, C, on),
    property(C, green, colour),
    class(C, object).
```

Subject grouping results in verb phrase coordination and removing type information results in more compact definite descriptions as shown in our target explanation at the beginning of Section 3. Note that reprocessing of this explanation by our bi-directional grammar results in a semantically equivalent rule.

3.3 Generating Definite Descriptions

During generation, the bi-directional grammar stores all the accessible antecedents and generates minimal definite descriptions on the fly. However, the grammar would generate an ambiguous verbalisation if we had a rule where the object in the head does not have a unique object in the body to link to after reconstruction and aggregation. To avoid this kind of an ambiguity, we add a variable to such an underspecified rule that allows us to distinguish between objects in an explicit way on the surface level of an explanation. For the learning of a house concept, this kind of an ambiguity occurs if we do not use type information for objects. Therefore, we add a variable to resolve the ambiguity (see Listing 4).

Now the following unambiguous verbalisation can be generated that introduces an indefinite noun phrase with a variable *an object A* as antecedent for the definite description *the object A* that occurs in the consequent of the sentence:

> *If an object A contains a blue object and contains a green object and the blue object is located on the green object then the object A is of type house.*

Listing 4: A sample rule after adding variables

```
class(A, object), variable(A, 'A'),
type(A, house) :-
    class(A, object),
    variable(A, 'A'), ...
```

4 Evaluation

We evaluate our explanation generation method in two ways: (1) we check if a generated explanation corresponds to a minimal and correct description of the image information, and (2) we check if the bi-directional grammar correctly works in both directions. For the first evaluation, we check whether an explanation is minimal or not by testing if the explanation meets our aggregation criteria. We check the correctness of an explanation by matching the literals used for the verbalisation with the corresponding literals for the image. For the second evaluation, we check the bi-directional grammar via a technique that is known as semantic round-tripping (Hossain and Schwitter, 2020); basically, we keep the formal representation *R1* for an explanation, feed that explanation again to the bi-directional grammar, generate a formal representation *R2*, and then compare if *R1* and *R2* are semantically equivalent.

In addition to our house concept dataset, we have employed the tower concept dataset and used single relation learning to evaluate our explanation generation method. This additional dataset is also used in the LIME-Aleph system to illustrate their method. For the learning of a tower concept, an image consists of three square objects of different colours and we say that the image represents a tower concept if one square is on the top of another square without the repetition of objects with the same colour. For single relation learning, we say that an image represents the *left of* relation if a green square is on the left side of a blue square. We used 1000 images from each dataset for the evaluation. For each image, the explanation is generated and evaluated using the above-mentioned technique. We have found that all the explanations for tower concept and for single relation learning

are correct while for house concept learning 999 explanations are correct. This gives us an accuracy of 100%, 100% and 99.9%, respectively.

5 Conclusion

In this paper, we have introduced a linguistic extension to HESIP, a hybrid explanation system, that combines sub-symbolic and symbolic representations for image predictions. This linguistic extension uses a bi-directional logic grammar to generate explanations in a CNL. The sub-symbolic component of HESIP makes a prediction that results in a probabilistic rule in the symbolic component of the system. The resulting rule is reordered according to linguistic principles, redundant information is aggregated, and possible ambiguities are resolved, before the rule is processed by the grammar. The output of the grammar is an unambiguous explanation of the prediction. If this explanation is not correct, then the user can modify the explanation and feed it back to HESIP.

The advantage of HESIP over the LIME-Aleph system is that it employs an object detection model to find objects in the images and uses an ontology to represent image information. The novelty of our hybrid approach to machine learning is that it allows us to generate explanations that are human-understandable as well as machine-processable; and it can be customised for other prediction tasks. HESIP can be used in any real-world application of image prediction where the images in the dataset have relations between objects. These relations can then be used in the probabilistic rules to explain the image predictions. Currently, we are investigating how HESIP can be extended and used to learn concepts from different parts of objects using the PASCAL-Part (Chen et al., 2014) dataset.

References

Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. 2014. Detect what you can: Detecting and representing objects using holistic models and body parts. In *Proc. CVPR'14*, pages 1971–1978.

Hercules Dalianis and Eduard Hovy. 1993. Aggregation in natural language generation. In *EWNLG'93*, pages 88–105. Springer.

Artur d'Avila Garcez and Luis C. Lamb. 2020. Neurosymbolic AI: The 3rd Wave. *arXiv preprint arXiv:2012.05876*.

David Gunning. 2017. Explainable artificial intelligence (XAI). *Defense Advanced Research Projects Agency (DARPA)*.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *IEEE ICCV'17*, pages 2980–2988.

Bayzid Ashik Hossain and Rolf Schwitter. 2020. Semantic round-tripping in conceptual modelling using restricted natural language. In *Australasian Database Conference*, pages 3–15. Springer.

Eleni Ilkou and Maria Koutraki. 2020. Symbolic Vs Sub-symbolic AI Methods: Friends or Enemies? In *CIKM (Workshops)*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS'12*, pages 1097–1105.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature*, 521(7553):436.

Stephen H. Muggleton, Ute Schmid, Christina Zeller, Alireza Tamaddoni-Nezhad, and Tarek Besold. 2018. Ultra-Strong Machine Learning: comprehensibility of programs learned with ILP. *Machine Learning*, 107(7):1119–1140.

Johannes Rabold, Hannah Deininger, Michael Siebers, and Ute Schmid. 2019. Enriching visual with verbal explanations for relational concepts–combining LIME with Aleph. In *ECML PKDD'19*, pages 180–192. Springer.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Why should I trust you?: Explaining the predictions of any classifier. In *ACM SIGKDD ICKDD'16*, pages 1135–1144. ACM.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. *In AAAI'18*, 32(1).

Fabrizio Riguzzi and Damiano Azzolini. 2020. cplint Manual. *SWI-Prolog Version*. Retrieved May 14, 2021 from `http://friguzzi.github.io/cplint/_build/latex/cplint.pdf`.

Rolf Schwitter. 2018. Specifying and verbalising answer set programs in controlled natural language. *Theory and Practice of Logic Programming*, 18(3-4):691–705.

Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe. 2004. Logic programs with annotated disjunctions. In *ICLP'04*, pages 431–445. Springer.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. 2019. Detectron2. Retrieved May 14, 2021 from `https://github.com/facebookresearch/detectron2`.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering*. DOI: 10.1109/TKDE.2020.2981333.

Toward a Reference Architecture for Traceability in SBVR-based Systems

Lloyd Rutledge
Open University of the Netherlands
Heerlen, The Netherlands
Lloyd.Rutledge@ou.nl

Rudy Italiaander
Tax and Customs Administration
of The Netherlands
Utrecht, The Netherlands

Abstract

We present an initial reference architecture for traceability in SBVR-based systems. It facilitates rule-based development that lets end users trace interface behavior back to the human decision points that lead to it. This closes a feedback loop, which facilitates agile development. The architecture is based on Web standards to generalize comparison and implementation.

It begins with the human process of linking document excerpts to the SBVR code that defines them. The next step transforms SBVR into computer code that implements it. Reasoners then form conclusions by applying the rules to data. They can also provide rudimentary explanations for these conclusions. The system then provides an end-user interface to all this rule-derived information. The core challenge here is maintaining the data needed to trace back through these layers, so end-user feedback can improve the entire development process.

1 Introduction

Semantics of Business Vocabulary and Business Rules (SBVR) provides a controlled natural language (CNL) for data models and business rules (Group, 2006). It can be mapped to more easily readable equivalents such as RuleSpeak (Spreeuwenberg and Healy, 2010). Such CNLs provide a step between stakeholders and programmers that helps designers communicate with both.

Rule-based systems constrain users to follow rules but often cannot explain where the rules come from or why they exist. This is because development of such systems often follows a sequential, or waterfall, approach. Analysis of legal text or policy documents leads to descriptions in SBVR. Other people then write program code for this SBVR. However, nothing in the code, and thus nothing in the resulting system interface, necessarily leads back to its source in SBVR or further back to the source documentation. The original motivation for the rules becomes effectively forgotten. End users end up several layers of one-way traffic away from understanding or potentially influencing change in the documents and discussions that form the rules they must follow.

Traceability in rapid prototypes of rule-based systems adds resilience to the process of forming laws and implementing them (Ausems et al., 2021). However, the mapping from human-readable business rules to computer-processed logic on the Semantic Web is complex (Spreeuwenberg and Gerrits, 2006). A reference architecture could help implement traceability in this complex mapping.

We propose forming a reference architecture in which this software development for business rule systems is less waterfall and more agile. It provides end users with transparency to the origins of rules in their systems so they can take part in their longer-term maintenance. In addition, development can then more easily include end-user representatives who provide short-term feedback over the effectiveness of both the documented rules and how the SBVR rules and resulting business systems implement them. Problems involving mismatches between high-level agreements and how well end-users of the system can appreciate and execute them can then be detected and addressed earlier and more often.

2 Reference Architecture

Fig. 1 illustrates our reference architecture. It has three broader components. First is the *development* of the specifications and code by people. Then comes the *automation*, which compiles the code into the business system. Finally, automation provides the system *interface* that the *end user* interacts with. The first step in development is *discussion* between human stakeholders, which results in *documents*, such as legislation or contracts, that define agreements reached. An *analysis* of the documents provides *annotation* of it, which shows the

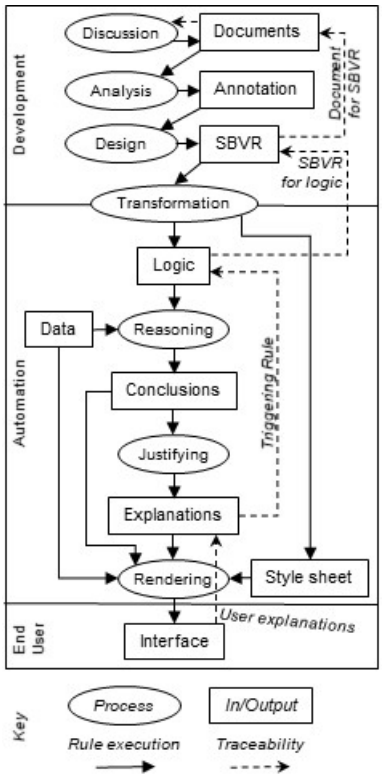


Figure 1: Our reference architecture for traceability in SBVR-based systems.

phrases for primary components in the system's data model and rules. The next process is the *design* of *SBVR* code for the model and rules.

With the SBVR code established, the *automation* of it is ready to start. The *transformation* of human-readable SBVR to machine-processable code can be human, automated or a combination of both. The resulting *logic* code defines the model and rules in a format a computer can process. The *reasoning* process applies the logic to the *data* in the system to generate *conclusions*. Conclusions can infer additional data, or find violations that the given data triggers. Core to traceability is automatically *justifying* these conclusions to provide *explanation* of them to the end-user.

The results of this automated processing need, of course, to be presented to the end user. The transformation of SBVR into system code includes not just the logic for data processing but also the *style* of this presentation. The *rendering* step processes this style code to determining how the data, conclusions and explanations should appear in the *interface*. This interface should not only let the user see all this information, but also trace back from it to the SBVR code and fragments of the documents from which the information originates.

3 Related Work

The software Cognitatie supports annotation of documents for conversion into knowledge models, including rules, placing it in the development block of our architecture (PNA Group). The tool RuleXpress processes business rules against a data set, functioning as the automation block (RuleArts). The Web service s2o[1] is a converter from SBVR to the Semantic Web ontology language OWL-2, thus automating the transformation process (Karpovic et al., 2014). Also on the Semantic Web, the ontology editor Protégé provides all automation processes, from reasoning on logic and data, to providing explanations (Musen and Protégé, 2015).

The Fresnel vocabulary for Semantic Web browsers is a format for style sheets mapping Semantic Web data to end-user interfaces to it (Pietriga et al., 2006). In earlier work, we developed software for generating Fresnel code for any given ontology (Rutledge et al., 2016).

HTML provides, perhaps obviously, a well applicable standard for source document descriptions of rules. We propose also having HTML browsers in the end user interface of our architecture provide direct user access to these source documents. For the system between rule documents and end users, the Semantic Web provides standards for data, its structure and rule-based logic applied to it. HTML and the Semantic Web both use URLs, which allows processing source document excerpts like any other Semantic Web concepts. Davis gives an overview of different ways to annotate HTML documents in the context of CNLs (Davis, 2013). Some techniques do not require editing of the annotated documents. The text to be annotated can be both CNL and the source document text they encode.

The RACE Reasoner for Attempto Controlled English processes input CNL logic and facts and outputs conclusions and reasoning explanations in CNL as well (Fuchs et al., 2008). The reference architecture we propose here models much of this processing, including processing CNL text to form conclusions. One difference is that our architecture does not explicitly account for generating explanations of conclusions in CNL. What the architecture does do here is provide links to the source CNL and document texts that defines the rules involved in a reasoning. As such, either the user can navigate to those document sources, or a natural language

[1] https://s2o.isd.ktu.lt/

generator can use these links to get the information needed to provide a readable explanation.

The course Rule-based Design that we teach at the Open University in the Netherlands uses the business rule reasoner Ampersand, which specializes in legal reasoning (Joosten, 2017). While Ampersand has no automatic generation of explanations behind its reasoning as RACE does, the Ampersand syntax provides the human editor some constructs for traceability (AmpersandTarski GitBook project). One is the RULE construct, which provides template-based explanation text, which the interface shows the user when the given rule is violated. This template text has placeholders for labels of specified components of the rule. In addition, the PURPOSE construct offers a human-readable citation of the document source for a rule or component of the data model. We propose here a PURPOSE-like citation for ontology components, but one that is machine-readable, and can link to source CNL chunks as well as document fragments.

Much real-world data exists on the Semantic Web, which matters for both practical application and academic validation of research results. For example, in earlier work, we created large amounts of data for applying Semantic Web logic to in order to generate statistics about the efficiency of that logic (Italiaander, 2019). Including the Semantic Web in our rule-based system development architecture facilitates testing and analysis with large amounts of existing or synthetic data.

4 Illustrative Scenario

We illustrate our reference architecture by applying it in an example scenario. This scenario derives from the fictional business EU-Rent, with its text descriptions and SBVR code (Object Management Group, 2016). In particular, we use a rule in EU-Rent's section G.6.7 "Car Movements". This EU-Rent section starts with the text "Car movements meet the business requirement that a car of a given group has to be moved between branches". We treat this as source text in our reference architecture. EU-Rent then provides the following SBVR rule it derives from that text: "*Necessity: Each car movement includes exactly one receiving branch*".

We convert this SBVR rule into machine-processable logic standard OWL with the tool s2o (Karpovic et al., 2014). Its equivalent in s2o's SBVR dialect that this scenario puts in s2o's rules field is "*It is necessary that car_movement has_receiving_branch exactly 1 branch*". The conversion's output is OWL code that makes car movements a subclass of the class of things with exactly one assignment for the property has_receiving_branch. The OWL constructs it uses are restrictions and qualified cardinality. This code corresponds to the logic code in the architecture.

A trigger for this rule is a car movement that is assigned to more than one different branch, resulting in an OWL inconsistency. In OWL, an inconsistency is a collection of facts and rules that cannot all be true. Fig. 2 shows a display from Protégé for such an inconsistency in our scenario. The top right of this display shows these two conflicting receiving branch assignments in red lettering, which in Protégé indicates an inconsistency. In the reference architecture, this shows a conclusion resulting from Protégé's OWL-defined reasoning.

When recognizing such an inconsistency, Protégé announces it to the user and then lets the user request an explanation for it. The explanation that Protégé then provides for our scenario appears in the lower part of Fig. 2. In this display under "Explanation 1", the first line shows the OWL code for the rule that is triggered. The other lines show the data that collectively triggers this rule. What traceability demands here is that the end-user be able to browse from the top line back to the SBVR code that derived it, and then back further to the document text that SBVR code defines.

5 Traceability

The solid arrows in the reference architecture diagram in Fig. 1 show the traditional view without traceability: we generate systems from rules, but can then easily forget the rules we generated the systems from. The dotted arrows show the primary places where traceability needs implementing. Each has a label describing which layer of traceability it provides. We describe them here in the order in which tracing happens here: from lower to higher in the diagram.

User explanations are the presentation of the explanations behind logical conclusions that appear in the interface for the end user. First, we should note that the process of justifying conclusions to provide explanations for them provides a layer of traceability. Logic systems such as Protégé implemented justification later than the simple display of conclusions, as it is a substantial technical challenge beyond generating the conclusions. Fig. 2

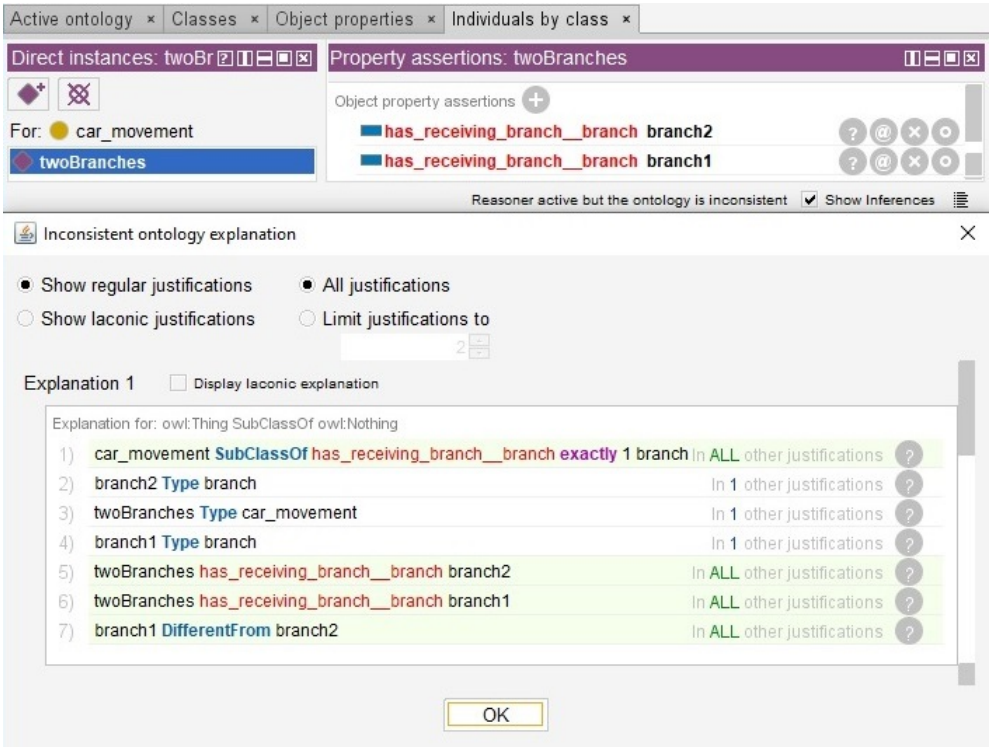


Figure 2: Protégé display of an inconsistency with its explanation.

shows how, in our scenario, Protégé informs the user that an inconsistency has occurred and which data it involves. Protégé shows an explanation only when and if the user requests it.

As this demo shows, this aspect of traceability is substantially implemented in Semantic Web systems. However, most end users cannot understand it, as it is an unstructured collection of all the data involved in forming the logical conclusion. It does have all the relevant information for an explanation, and software could process it for a CNL explanation, such as RACE does for its internal logic. However, the focus in this reference architecture is on how to access this information instead of how to present it understandably. In addition, our architecture focuses more on traceability as accessing the documented origins behind the logical rule applied in a situation.

The next dotted arrow, "*triggering rule*", in the architecture diagram goes from explanations back up to logic program code. This is where our architecture focuses on human-accessible traceability instead of the automated generation of human-understandable text, as other research does. Instead of processing all of the triples of an explanation into text, we propose focusing on the information regarding the rule that is triggered. That rule can then, in turn, lead to the CNL text and original document text that people wrote earlier to describe that rule. The other triples in the explanation data describe mainly the data that triggers the rule, instead of the triggered rule itself. The challenge here is to find the code for the triggered rule from all the data for an explanation.

In Fig. 2's scenario, the rule triggered is in the first line of the explanation, especially where it says "exactly 1". This line refers to the code for an OWL restriction, which has Semantic Web-defined properties that define the restriction. One property assignment declares the type of cardinality as exactly one. Another assigns it to the property "has receiving branch". This restriction resource is then assigned as a superclass of car movements. This is how the restriction defines that all movements much have exactly one receiving branch.

The question is then: where does this OWL-code come from? The "*SBVR for logic*" dotted arrow leads from the machine-processed logic code back up to that rule's definition in layperson-readable SBVR text. A platform providing this function would need to support links from chunks of logic code, such as OWL, to fragments of SBVR text. HTML could encode these fragments of SBVR text as anchors, giving each a URL. Then in the au-

tomation layer, Semantic Web code can associate these SBVR fragment URLs with the Semantic Web-defined logical constructions that implement them. What the end user then sees in the interface is navigable web browser links from the display of the triggered rule in the explanation to the corresponding piece of SBVR. In our example, this would be an additional property linking the restriction to the fragment of SBVR code defining it. The value would simply be a URL with a fragment identifier linked to the relevant portion of SBVR code in its online document.

The next step, "*document for SBVR*", lets the user go from such a segment of SBVR code to the extract of documentation that is its source. Again, this could be a URL leading from that SBVR fragment to the fragment of the other document providing its original definition. If the SBVR is in HTML in order to link it to the Semantic Web, then this HTML can also give the user a hyperlink from SBVR to the corresponding portions of the HTML-defined source document. The end user can then browse from reasoned conclusions through the triggered rule in an explanation, back to the rule's SBVR source, and then to the document text. This enables the final step in rule traceability: discussing the document text with those who formed it, perhaps in order to improve it, which in turn improves how well system end users can understand and carry out those rules. With such a reference architecture for traceability, we aim to help developers of software for this architecture's processes craft exchange formats that let all components of the broader platform exchange all information needed to provide a traceable whole.

6 Acknowledgements

We thank Mariette Lokin for her helpful comments and suggestions about this architecture and paper.

References

AmpersandTarski GitBook project. Documentation of Ampersand. https://ampersandtarski.gitbook.io/documentation/. Accessed: 2021-07-05.

Anouschka Ausems, John Bulles, and Mariette Lokin. 2021. *Wetsanalyse voor een werkbare uitvoering van wetgeving met ICT*. Boom Juridische Uitgevers.

Brian Patrick Davis. 2013. *On Applying Controlled Natural Languages for Ontology Authoring and Semantic Annotation*. Ph.D. thesis, National University of Ireland Galway.

Norbert E Fuchs, Kaarel Kaljurand, and Tobias Kuhn. 2008. Attempto controlled english for knowledge representation. In *Reasoning Web*, pages 104–124. Springer.

Object Management Group. 2006. Semantics of Business Vocabulary and Business Rules (SBVR). *OMG Specification*.

Rudy Italiaander. 2019. AgentRole, TimeInstant en InverseOf Ontology Design Patterns voor efficiëntere afleidingen van beweerde data. Master's thesis, Open University of the Netherlands, Heerlen, The Netherlands.

Stef Joosten. 2017. Software Development in Relation Algebra with Ampersand. In *Relational and Algebraic Methods in Computer Science*, pages 177–192, Cham. Springer International Publishing.

Jaroslav Karpovic, Gintare Krisciuniene, Linas Ablonskis, and L. Nemuraite. 2014. The Comprehensive Mapping of Semantics of Business Vocabulary and Business Rules (SBVR) to OWL 2 Ontologies. *Inf. Technol. Control.*, 43:289–302.

Mark A. Musen and Team Protégé. 2015. The Protégé Project: A Look Back and a Look Forward. *AI matters*, 1(4):4–12.

Object Management Group. 2016. Semantics of Business Vocabulary and Business Rules (SBVR), Appendix G - EU-Rent Example.

Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. 2006. Fresnel: A browser-independent presentation vocabulary for RDF. In *International Semantic Web Conference*, pages 158–171. Springer.

PNA Group. Cognitatie. http://www.cognitatie.nl/. Accessed: 2021-07-05.

RuleArts. RuleXpress. http://www.rulearts.com/RuleXpress. Accessed: 2021-07-05.

Lloyd Rutledge, Thomas Brenninkmeijer, Tim Zwanenberg, Joop van de Heijning, Alex Mekkering, J. N. Theunissen, and Rik Bos. 2016. From Ontology to Semantic Wiki – Designing Annotation and Browse Interfaces for Given Ontologies. In *Semantic Web Collaborative Spaces*, pages 53–72, Cham. Springer International Publishing.

Silvie Spreeuwenberg and Rik Gerrits. 2006. *Business Rules in the Semantic Web, Are There Any or Are They Different?*, pages 152–163. Springer Berlin Heidelberg, Berlin, Heidelberg.

Silvie Spreeuwenberg and Keri Anderson Healy. 2010. SBVR's Approach to Controlled Natural Language. In *Controlled Natural Language*, pages 155–169, Berlin, Heidelberg. Springer Berlin Heidelberg.

Facilitating the application of Controlled Natural Language (CNL) to standardize communication in logistics and supply chain management

Joel Cedric Lengeling
Chair of Logistics
Technical University of Berlin
lengeling@logistik.tu-berlin.de
alternatively joel.cedric@lengeling.eu

Abstract

Communication protocols allow to standardize communication. They are typically implemented to standardize the exchange of messages in the area of information systems. Nevertheless, by applying Controlled Natural Language (CNL), it is possible to implement communication protocols that allow both; participation by humans, while still enabling information systems to accurately and efficiently process the very same messages. Thus allowing computers and humans to communicate in unison. Here an artifact that allows applying formal CNLs for communication in the domain of logistics is presented.

1 Introduction

It is a truism: communication plays an important role in logistics. Delays, production schedules, missing spare parts ... and the information thereof has to be communicated. Although communication plays such an important role, it is surprisingly neither always standardized, nor is it - partly result of that missing standardization - always automated and misunderstandings are commonplace. This contribution aims to mitigate that situation by promoting the application of Controlled Natural Languages (CNLs) in logistics communication introducing an artifact.

To assess this artifact some observations on the logistics industry and some observation on, more or less, typical communication situations in that industry are presented. Part of those observations are based on interviews conducted with some arbitrarily chosen experts in the field. In addition some casual conversations with industry veterans do influence that picture: logistics is an highly competitive market, profit margins are low, and the fear for competition is commonplace. Logistics business arrangements are often only short term and the small organizations forming a large part of that market are rarely big enough to allow for an "automation" department. The average employee is typically non-academic and the environment is multi-lingual and multi-cultural. In addition the following observations on communication in the domain of logistics have been made: communication may either be scheduled or non-scheduled, it may often be granular, it might often reoccur regularly, and the variance between occurrences may often be rather low.

Some reasons identified by the experts why communication is typically not yet fully automated have been: 1) the incompatibility of information systems, 2) the sizes of participating organizations, 3) the mentality towards information transparency, 4) the mentality towards digitalization, and 5) the cost of implementation. As a result opportunities for digitalization are missed.

Standardizing communication in the domain of logistics applying CNLs may potentially improve that situation. Kuhn mentions that there is no generally agreed-upon definition of CNL and describes the insight that "CNLs can be conceptually located somewhere in the gray area between natural languages on the one end and formal languages on the other"(cf. Kuhn, 2014). Here we work with CNLs that are on the formal language end. ISO/IEC/IEEE 24765:2017(E) defines a formal language rather conversational as "language whose rules are explicitly established prior to its use"(24765, 2017 p. 188). Mateescu defined formal languages less conversational applying Σ and Σ^* (cf. Mateescu and Salomaa, 1997 pp. 10-11). The alphabet Σ is a finite nonempty set of which the elements are called letters or symbols and Σ^* is a set of all words or strings consisting of zero or more letters of Σ. Subsets, finite or infinite, of Σ^* are referred to as formal languages over Σ. We work with CNLs that are formal languages that apply words

or strings from one specific natural language - the base language - in such a way, that the essence of texts written in that CNL may be understood by the average employee that understands the base language from which the words or strings originate. In formal language theory a grammar of a language is a mechanism that allows the production of sets of strings in that language (cf. Harrison, 1978 p. 13). Essentially, formal languages are described by their syntax. The semantics of a formal language is, at least in computer science textbooks, rarely discussed. The semantics of a formal CNL is here taken from the base language that provided the words or the strings to that CNL. Extended Backus–Naur form (EBNF), an extension of Backus–Naur form (BNF), allows to express the grammar of the formal languages in mind. *A Restricted English for Constructing Ontologies (RECON)* is actually an example of a CNL that has been expressed by BNF (Barkmeyer and Mattas, 2012). Standardizing communication applying a CNL of which the grammar is e.g. expressed applying EBNF would allow the automation of that communication, while at the same time allowing "participation" of a non-academic workforce speaking the base language of the CNL. Nevertheless, as of today, standardizing communication applying CNLs is rarely discussed in both the logistics scientific community as well as in the logistics industry. Here an attempt resulting in an artifact that may lead to a more often application of CNLs in logistics is presented. This attempt is conducted from the viewpoint of a software engineer. The goal is to present an artifact that allows simple and flexible application while being rather maintainable and independent of other systems.

2 A limited literature review

A limited search was conducted in three research databases: A) *Business Source Complete (via EBSCO Host)*, B) *Web of Science Core Collection*, and C) *IEEE Xplore Digital Library* during the summer of 2021. The following search string has been applied: *("controlled natural language" OR "cnl" OR "domain specific language" OR "dsl" OR "formalized language" OR "processable language") AND ("logistics" OR "scm" OR "supply chain" OR "operations management")*. By taking that approach 58 results in A), 5 results in B), and 32 results in C) have been identified. After filtering for duplicates, non-scientific publications, and publications that have been considered off-topic, 19 publications remained. Of those 19 publications, 17 mentioned DSL, 1 mentioned CNL, and 1 mentioned both, DSL and CNL. Following Deursen a DSL is "a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain"(Van Deursen et al., 2000). From the viewpoint taken here 1) a CNL may be also classified as a DSL, if the CNL is a formal language, and 2) a DSL may be also classified as CNL, if the DSL incorporates natural language and is expressive enough to allow usage by anybody fluent in that natural language. Thus some languages may be classified as both, CNL and DSL. Of the 17 contributions mentioning DSLs, 1 contribution introduces a DSL which at least from our perspective may also be classified as CNL. That language is called *Logistics Task Language* and it allows "to describe intra-logistics material flow processes" (Detzner et al., 2019). The identified contribution that mentions CNL, specifically does mention CNL as part of a system that supports manual assembly planning (Manns et al., 2018). Nevertheless, from the perspective taken here, both do not introduce a generalized system for logistics communication. The contribution which mentions both DSL and CNL does this, by discussing a use case related to logistics in the context of ProjectIT, an approach and tool for requirements engineering (Da Silva et al., 2007). Nevertheless, from the perspective taken here, this approach does not seem suitable to generalize communication in logistics as described before. The earlier mentioned language RECON was not found by applying the aforementioned search strings showing the limit reach of the literature review. Nevertheless RECON may still be applicable in logistics. At the end of this short overview, it might be important to point out, that there may be more CNLs applied in the actual field that are not subject to scientific publications. One example coming to mind are Pick-By-Voice systems that are most likely implemented applying CNLs, but nevertheless no publication discussing how to implement such a system was found within the limited search here.

3 Applying CNL to standardize communication in logistics

To simplify the task at hand this work builds on a flexible and comprehensible model of a *communication protocol* out of the domain of computer science presented by Holzmann. The advantage of that model is, that it allows a rather flexible approach to implement standardization without going into as much detail as many other technical standards in the field do; e.g. for application in logistics it suffices to specify that messages are transmitted by (e-)mail including an (e-)mail address instead of specifying the communication down to the bit level over the wire. Another benefit of that approach is it's ability to just describe the exchange of one message as part of one protocol. A complete message exchange may be described applying multiple instances of one protocol. Following Holzmann a protocol is the sum of all rules, all formats, and all procedures that have been agreed upon, between at least two computers in order to communicate (cf. Holzmann, 1991 pp. 19-21). Here, that definition is slightly extended, to also cover communication between either two humans or an human and a computer. A message that contains the content of the communication may be created by applying the rules, formats, and procedures specified as part of that protocol. Following Holzmann a protocol specification consists out of five distinct parts: 1) the *service* to be provided by the protocol, 2) the *assumptions* about the environment in which the protocol is executed, 3) the *vocabulary* of messages used to implement the protocol, 4) the *encoding* (format) of each message in the vocabulary, and 5) the *procedure rules* guarding the consistency of message exchanges.

The *Design Science Research Process Model* (DSRPM) as introduced by Vaishnavi and Kuechler (cf. Vaishnavi and Kuechler, 2015 pp. 14-18) has been applied to develop the artifact. This model comprises five process steps which are frequently iteratively performed: *awareness of problem*, *suggestion*, *development*, *evaluation*, and *conclusion*. The awareness of the problem described before originated from casual conversations with industry professionals. As an initial solution, a system that allows end user development of simple problem specific programming languages for communication that are designed applying everyday language and that thus may be understood by the average employee, has been first suggested in a casual setting to an industry professional during the summer of 2020. Since then, multiple DSRPM iterations have been conducted. The artifact version discussed here was subject of more extensive and structured interviews with industry experts during the spring of 2021. For those interviews 9 experts have been reached out to, 7 responded, and 5 confirmed, that they were willing to allocate the necessary time. The conducted interviews took up to two hours and had 3 stages: 1) explanation of the theory, 2) presentation of tangible examples from logistics, and 3) a detailed discussion. In order to receive comparable results such a discussion was guided by a pre-prepared presentation containing questions and discussion points that dealt with the applicability of the artifact in logistics, communication in logistics, and the technical implementation of the artifact. A more extensive discussion of the conducted interviews will be published in a more extensive contribution. 1 expert is the CEO of an automotive supplier, 2 experts have a leadership role in departments that are responsible for business innovations; 1 at a global logistics service provider and 1 at a global manufacturing cooperation. 1 expert is a consultant currently working on a communication automation project in logistics, and the final expert is a purchasing agent at a pharmaceutical trading company. This diverse field was approached to gain rather diverse feedbacks.

The artifact is supposed to be applicable for the standardization of communication on the operational level, the 'day-to-day' communication, between organizational dyads in logistics. Thus here, we are neither dealing with communication within one organization, nor with communication within a higher level polyad. Following the psychologist Tomasello, the three cooperative social motivations for communication among humans *requesting*, *informing*, and *sharing* do exist (Tomasello, 2008 pp. 82-88). The motivation for communication that may be standardized applying the artifact should be *requesting* or *informing*. Finally to allow for standardization the communication should be either scheduled or non-scheduled and in addition, it should be granular, it should reoccur regularly, and it should have a low variance between occurrences.

The artifact builds on the previously discussed model by Holzmann. The experts were asked if that model seems applicable to logistics and the tendency of the experts has been, that the model is plausible. Protocols that originate applying this artifact may be easily modified due to the limited number of partners. A new version of the protocol is created when it is modified. The information that is needed for each one of the five distinct parts of a protocol for the domain of logistics has been worked out during the iterations of the DSRPM. For each distinct part the experts have been asked to provide feedback. Eventually the following information requirements have been identified for 1) service, 2) assumptions, and 5) procedure rules: **1) service**: a) the version and name of the protocol, b) the motivation for the communication, c) the responsible contact persons at each organization and the two participating organizations, d) a reference to the base language of the applied CNL, and, if applicable, a list of equivalent protocols that apply different CNLs, and e) some keywords allowing to identify and find the protocol in a protocol management system or database; **2) assumptions**: a) how messages applying the protocol are communicated, b) the encoding technologies of the CNL (how the grammar is defined, e.g. (E)BNF, Grammatical Framework, ...) c) how messages applying the protocol are created (by human/ by machine and the solution for the writability problem), and d) how messages applying the protocol are processed; **5) procedure rules**: a) when, which organization initiates the communication, and b) if there is a follow up to the communication. The **3) Vocabulary** is a plain language area for: comments, feedback, and explanations. According to the International Plain Language Federation, communication is in plain language "if its wording, structure, and design are so clear that the intended readers can easily find what they need, understand what they find, and use that information"(Onl). The *vocabulary* is written for a neither technical nor academical audience containing all the information deemed necessary to apply the protocol in the field, helping to prevent errors and misunderstandings. The semantics of a formal CNL is here taken from the base language. Thus, due to e.g. homonyms or jargon misunderstandings are still possible. It is important that as part of the *vocabulary* semantics of such cases may be explained. During the specification of the protocol it is important to determine what needs to be part of the *vocabulary*. Nevertheless, should an error or misunderstanding occur, a new protocol version preventing such an occurrence in the future may be simply created. The *vocabulary* should aim to be as compact as possible and it shouldn't deter by seemingly looking to complex, e.g. by containing anything but plain language, or by looking to extensive, e.g. by being some kind of lexicon. The **4) Encoding** has to describe the syntax of the messages exchanged. Either a new case specific CNL may be expressed or an existing CNL, e.g. RECON, may be applied. If a new CNL is expressed the grammar of the CNL needs to be expressed; e.g. a syntax diagram may be depicted. If an existing CNL is applied an external link to the specification may be provided or the applied aspect of that CNL may be cited. The syntax should ensure that created messages allow identification of the applied protocol. Similarly to the *vocabulary*, the *encoding* should aim to be as compact as possible and it shouldn't deter by seemingly looking to complex or to extensive. More complex communication may be broken down into multiple smaller communication units, for which separate protocols with more case specific CNLs may be created. In addition a process chain diagram from the domain of logistics as depicted in Fig. 1 may be specified as part of the protocol specification if the communication is scheduled communication. Finally a diagram called 'stakeholder communication diagram' depicted in Fig. 2, developed, based on feedback provided during the interviews and inspired by so-called 'dialogue trees' (see Adams, 2010 p. 186), may also be part of the specification. The example diagram is part of the specification of protocol B (indicated by the red arrow). The Initial Stakeholder is able to communicate with Stakeholder 1 applying Protocol A and with Stakeholder 2 applying Protocol B. Stakeholder 2 is optionally - as indicated by the dashed arrow - able to respond back to the Initial Stakeholder applying Protocol C. Stakeholder 2 is also able to communicate to Stakeholder 3 - 'forwarding' the information - applying Protocol D.

4 Conclusion

The tendency of the expert opinion was, that there is a place for the presented artifact in logistics. Nevertheless, the tenor was, that adequate information systems are necessary to allow the application by an average employee. Such a

Figure 1: A process chain diagram depicting a communication process and the processes surrounding this process. Own depiction based on Baumgarten and Inga-Lena, 2000

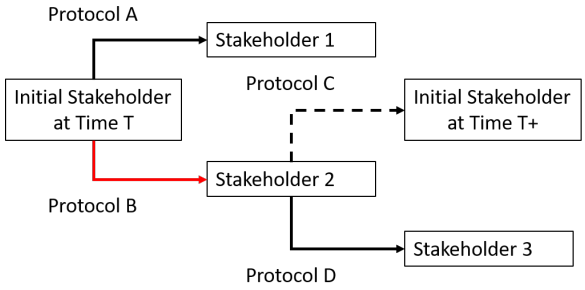


Figure 2: A stakeholder communication diagram.

system requires a protocol creation and versioning component. Protocols implemented applying the artifact should be integratable into systems that allow integration. Due to the limitation - only communication between dyads - and the flexibility of the encoding, modifications are easily implemented. The participating organizations are in charge of which information they share due to the same reason. Communication in a multi-lingual and multi-cultural environment should also be possible by creating equivalent protocols applying CNLs applying different base languages. To summarize, the artifact promotes the application of formal CNL in the domain of logistics.

There are multiple possible future research directions. More research into the application of the presented artifact in the field is required. This type of protocol discussed, may also be applicable in different domains. Additionally, research towards information systems, that would allow the application by the average employee, is needed. During the interviews multiple experts remarked, that it would be helpful, if the protocols are able to apply machine learning to further develop themselves. Thus dynamic protocols, that apply machine learning to further develop themselves, may also be an interesting topic for future research.

References

Iplf, plain language. www.iplfederation.org/plain-language/. Accessed: 2021-07-05.

ISO/IEC/IEEE 24765. 2017. Iso/iec/ieee international standard - systems and software engineering–vocabulary. *ISO/IEC/IEEE 24765:2017(E)*.

Ernest Adams. 2010. *Fundamentals of game design*, 2nd edition. Pearson Education.

Edward Barkmeyer and Andreas Mattas. 2012. *A Restricted English for Constructing Ontologies (RECON)*. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD.

Helmut Baumgarten and Darkow Inga-Lena. 2000. Prozesskettenmanagement als basis für logistikmanagement. In *Logistik-Management Strategien Konzepte Praxisbeispiele*, 1st edition, chapter 2.3.1. Springer-Verlag.

Alberto Rodrigues Da Silva, João Saraiva, David Ferreira, Rui Silva, and Carlos Videira. 2007. Integration of re and mde paradigms: the projectit approach and tools. *IET software*, 1(6):294–314.

Peter Detzner, Thomas Kirks, and Jana Jost. 2019. A novel task language for natural interaction in human-robot systems for warehouse logistics. In *2019 14th International Conference on Computer Science & Education (ICCSE)*, pages 725–730. IEEE.

Michael A. Harrison. 1978. *Introduction to Formal Language Theory*. Addison-Wesley Series In Computer Science. Addison-Wesley Publishing Company.

Gerard J Holzmann. 1991. *Design and validation of computer protocols*. Prentice hall Englewood Cliffs.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational linguistics*, 40(1):121–170.

Martin Manns, Klaus Fischer, Han Du, Philip Slusallek, and Kosmas Alexopoulos. 2018. A new approach to plan manual assembly. *International Journal of Computer Integrated Manufacturing*, 31(9):907–920.

Alexandru Mateescu and Arto Salomaa. 1997. Formal languages: an introduction and a synopsis. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*, pages 1–39. Springer-Verlag New York, Inc., New York, NY.

Michael Tomasello. 2008. *Origins of Human Communication*. MIT press.

Vijay K Vaishnavi and William Kuechler. 2015. *Design science research methods and patterns: innovating information and communication technology*, 2nd edition. CRC Press.

Arie Van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36.

A CNL-based Method for Detecting Disease Negation

Joan Byamugisha and **Nomonde Khalo**
IBM Research Africa
45 Juta Street, Braamfontein
Johannesburg, 2000, South Africa
joan.byamugisha@ibm.com and nomonde.khalo@ibm.com

Abstract

Negation detection is a key feature to the processing of biomedical text, and it involves two steps: identifying a medical term of interest in text and identifying that that medical term is mentioned as absent. However, processing biomedical text is made complex by the presence of medical jargon that typically requires custom systems, and detecting negation is complicated further because the representation of negation in natural language varies according to the grammar used. We investigated the use of a CNL with a general-purpose semantic parser to detect negation. Our CNL was created by representing medical terms as their semantic types and restricting the definition of the expression of negation. Through this method, we identified three kinds of negation–explicit negation, implicit negation, and explicit implicit negation. A pilot evaluation of our method on a sample of radiology reports achieved an F1 score of 0.99 on the sentences that could be parsed.

1 Introduction

Natural language (also referred to as free text or narrative) is the most wide-spread, comprehensive, and convenient medium for healthcare personnel to present medical information, for example, in patient progress notes, radiology and pathology reports, and discharge summaries (Wang et al., 2018). Narrative patient reports contain several medical concepts (naming entities such as body parts, drugs, symptoms, diseases, medical tests, and treatments) and the relations between the concepts (Wang et al., 2018). Identifying and distinguishing among different entities is the basis of biomedical NLP tools for text classification, named entity recognition, and text summarization, section detection, and negation detection, among others. In this paper, we focus on the task of disease negation detection, an important aspect in biomedical NLP, which involves two steps: identifying a medical term of interest in text and identifying that that medical term is mentioned as absent (Chapman et al., 2001). Both these tasks are complicated by the nature of medical jargon and the variation in the representation of negation in natural language.

Current approaches taken to solve the problem of identifying negated medical terms can be categorized as syntactic-based, ontology-based, and corpus-based. Syntactic-based systems typically rely on the use of custom regular expressions for pattern matching and to combine grammar parsing with standard expression matching (Huang and Lowe, 2007). Ontology-based systems, such as (Elkin et al., 2005), apply the knowledge from an ontology to standardize the representation of medical terms, thus enabling their automatic interpretation during negation detection. Corpus-based systems use machine learning algorithms to learn the scope of negation and treat negation detection as a classification task (Slater et al., 2021). The limitations of these approaches include: for syntactic-based systems, limited coverage due to being restricted to the syntax defined in a regular expression, offering little or no contextualization of the syntax to the semantics in the text, and tedious maintenance; for ontology-based systems, the level of semantic contextualization provided is limited and lacks broader coverage of the entire text; and for corpus-based systems, they require large amounts of data that are not readily available in the healthcare domain; and the black-box nature of machine learning algorithms makes it impossible to assess what aspects of negation they are learning.

On the other hand, there exist very efficient and highly accurate general-purpose semantic parsers that can be used to detect negation. The problem here is that the presence of medical jargon in natural language text increases the complexity and ambiguity already inherent in natural language, and renders attempts at using general-purpose parsers unreliable, as they result in inaccurate semantic

representations. For example, parsing sentences with medical jargon using the ACE parser (Packard, 2013) produces results with semantic categories identified as 'unknown'. As the source of the complexity is the presence of medical jargon, we hypothesized that reducing this complexity to a vocabulary that can be parsed by general-purpose semantic parsers can reduce the negation detection problem to that present in a domain-independent vocabulary, and enable the use of a general-purpose semantic parser to perform negation detection of disease entities. Our work contributes: (1) a method of defining a CNL by first looking at natural language and then restricting its lexicon and expression of negation; and (2) a pilot method, still to be evaluated comprehensively, for negation detection in a medical domain using a general-purpose English parser.

2 CNL-based Negation Detection

For the task of negation detection, we sought to limit the impact of medical jargon by converting medical text into a restricted version that can be parsed deterministically, CNL, using a general-purpose English parser. We used an efficient linguistic processor for Head-driven Phrase Structure Grammars (HPSGs), the Answer Constrained Engine (ACE) (Packard, 2013), which supports most modern computational linguistic features. A broad-coverage symbolic grammar of English–the English Resource Grammar (ERG) (Copestake and Flickinger, 2000), was used to parse the CNL and analyzed the Minimal recursion semantics (MRS) (Copestake et al., 2005) representations for signals of the negation of mentioned entities. The following sections present details on the materials, methods, and results of this work.

2.1 Materials

The data used in this investigation was obtained from the corpus of radiology reports from the Mimic CXR dataset (Johnson et al., 2019). We used a sample of 100 reports and considered only the sentences associated with the sections in a report which contain conclusions about the findings in a report. These sections are labeled as *FINDINGS*, *IMPRESSIONS*, or *CONCLUSIONS*, or their singular forms. Some reports do not have these sections, while others have at least one of these sections. From our sample of 100 reports, 92 were found to possess at least one of these sections, and from these reports, 345 sentences were obtained. These sentences were examined manually to remove any sentence that contained the after effects of report deidentification (such as '___ at ___ on ___.' and 'Analysis is performed in direct comparison with the next preceding similar study of ___.'). These were removed, resulting in a final dataset of 316 sentences.

A ground-truth dataset was created manually for these sentences. The criterion used when labeling the dataset was that, if there is at least one indicator of disease which is mentioned as present, then that sentence is labeled as *N* for 'not negated', otherwise, it is labeled as *Y*. The rationale behind this labeling scheme is that the purpose of negation detection is to identify patients who have at least one disease indicator as opposed to patients who have none. Therefore, sentences such as, 'Bilateral pleural effusions, severe pulmonary edema, cannot exclude pneumonia.' and 'The heart size is normal, but the pulmonary vasculature is still mildly engorged.' are labeled as *N* (not negated); while a sentence such as 'Heart size is enlarged but stable.' is labeled as *Y* (negation present). Of the 316 sentences in the dataset, 136 sentences were annotated as negating the mentioned disease indicators, while 180 were annotated as possessing at least one present disease indicator.

Our CNL was created by representing medical terms as their semantic types and restricting the definition of the expression of negation. Though simple, we ensured that the result possessed all the four properties by which a language can be regarded as a CNL: (1) it is based on exactly one natural language, its base language; (2) the more restrictive lexicon is the most important difference between it and its base language, and we restrict further its expression of negation; (3) it preserves most of the natural properties of its base language; and (4) it is explicitly and consciously defined (Kuhn, 2014). When constructing our CNL, we selected two semantic types–*Anatomy* and *Disease*–because we are focusing on detecting the negation of diseases. The *Anatomy* semantic type is required in order to contextualize disease mentions that are expressed through an anatomical region where a disease occurs. Creating the CNL requires identifying a medical term in text and then determining its semantic type. We relied on the knowledge in the Unified Medical Language System (UMLS) to determine whether a medical term represents a

disease or an anatomy, and we selected six terminologies in the 2020 release of the UMLS metathesaurus purposively so as to have a broad coverage with which to identify diseases and anatomies. For parsing with ACE (Packard, 2013), we used the English Resource Grammar (ERG) (Copestake and Flickinger, 2000) and selected MRS (Copestake et al., 2005) as the representations with which to analyze the results.

2.2 Methods

First, QuickUMLS (Soldaini and Goharian, 2016) was used on each sentence to extract medical concepts and their corresponding Concept Unique ID (CUI). 392 medical terms and their CUIs were extracted from 316 sentences. Next, the semantic type of an entity was determined by mapping a CUI to each of the five selected terminologies, which is possible because the UMLS Facilitates conceptual mappings among terminologies. If a mapping from a source terminology to a target terminology produces concept(s), then it implies that that concept is found in the target terminology, and is, therefore, of the semantic type represented by that terminology. Based on this criterion, of the 392 medical entities extracted, 62 entities representing anatomies and 64 entities representing diseases were found.

After this, creating a CNL of each sentence was done by replacing a medical term with the semantic type associated with it. For example, 'right rotator cuff' becomes *Anatomy* and 'interstitial edema' becomes *Disease*. Additionally, where multiples of the same semantic type are present in a sentence, they are numbered so as to differentiate them to the parser and maintain the semantics in a sentence. For example, 'There is no evidence of pneumothorax, pleural effusion, pulmonary edema, or pneumonia.' becomes, 'There is no evidence of Disease1, Disease2, Disease3, or Disease4.'. Through this process, medical jargon is reduced to representations of proper nouns that can be parsed using ACE. Finally, we applied two types of negation: explicit negation and implicit negation. Explicit negation is detected through the presence of negation markers and qualifiers in the MRS output. In MRS, explicit negation is represented with '*neg*'. Additionally, the quantifier for a noun, if found to be '*no*', semantically signifies that an entity is present zero times, hence, negation. For example, in the sentence 'There is no evidence of pneumothorax, pleural effusion, pulmonary edema, or pneumonia.', 'no' is a quantifier signifying zero 'evidence'; as opposed to, say, 'some evidence of' or 'only evidence of'. In our CNL, we restrict implicit negation as detected through a limited vocabulary. We extracted adjectives, nouns, and verbs from the MRS representations and identified semantic constructions that indicate the presence or absence of a disease. For the former, constructions such as 'present_a_1', 'indication_n_of', and 'worsen_v_cause' point to the presence of a disease; for the latter, constructions such as 'clear_a_of', 'normal_n_1', and 'rule_v_out' point to the absence of a disease.

2.3 Results

Of the 316 sentences with ground-truth negation values, 267 were parsed successfully with ACE, while 49 (15.51%) could not be parsed and had no MRS representations. We, therefore, present results obtained from the 267 sentences. 89 sentences (33.33%) were found to contain a conjunction, while 28 (10.49%) contained the explicit negation construct '*neg*' and 70 (26.22%) contained 'no' as a quantifier of diseases. Therefore, the total number of sentences with constructs associated with explicit negation was 98 (36.7%).

Of the 115 sentences annotated as negated in the ground-truth, 81 were identified correctly through explicit negation. The false negatives from using explicit negation only comprise sentences that either express a disease by referring to an anatomical region instead of a disease directly; or describe the absence of a disease without negating its presence explicitly, rather implicitly. Examples of sentences where a disease is negated by describing the affected anatomy are, 'The heart size remains normal as well as the thoracic aorta which follows the scoliotic curvature in its descending portion remains within normal limits.', and 'The cardiac, mediastinal and hilar contours appear stable.'. Cases where the presence of a disease is negated implicitly are, 'Since the prior exam, the lung volumes have improved.', and 'The right perihilar opacification and bilateral pleural effusions have resolved.'.

For implicit negation, we catered for two cases: (1) implicit negation either anatomically or through disease; and (2) negation of implicit negation. When investigating this, the following parts-of-speech were extracted: 155 adjectives, 166 nouns, and 91 verbs. Of these, 5 adjectives, 5 nouns, and 5 verbs were included in a vocabulary as signify-

ing the absence of a disease; while 3 adjectives, 9 nouns, and 7 verbs were included in a vocabulary as signifying the presence of a disease. Implicit negation considers two kinds of semantics–those that signify the presence of a disease and those that signify the absence of a disease. The vocabulary required to identify implicit negation is very small when compared to the parts-of-speech extracted, that is, 8 out of 155 adjectives, 14 out of 166 nouns, and 12 out of 91 verbs were necessary. Explicit implicit negation presents a situation of a double negative, and it, therefore, reverses the negation semantics of the parts-of-speech used to indicate the presence or absence of a disease. For example, in the sentence, 'The presence of a minimal left pleural effusion cannot be excluded.', the word 'exclude' that would have indicated the absence of a disease now indicates the presence of a disease because it is negated.

Our method detected negation implicitly and also checked for explicit implicit negation. Of the 115 sentences annotated as negated in the ground-truth, an extra 33 were identified correctly through implicit and explicit implicit negation; while 93 out of the 152 unnegated sentences were identified correctly through this method. The presence of conjunctions was used to identify 17 non-negated sentences correctly, and another 41 sentences were identified correctly as unnegated because they contained the terms signifying the presence of a disease. When considering the number of sentences that could be parsed by the ACE parser, then an F1 score of 0.99 is obtained. However, when considering the entire ground-truth, including the sentences that could not be parsed, then the F1 score is 0.84.

3 Conclusion

In this paper, we have presented a method of defining a CNL from natural language by restricting the lexicon of the CNL and restricting the definition of the expression of negation. The lexicon is restricted by representing disease and anatomical medical terms as their semantic types, allowing for the processing of medical text using general-purpose semantic parsers. The second restriction of our CNL is that negation can be expressed through explicit negation, implicit negation, and explicit implicit negation. We conducted a pilot study that shows that a high F1 score (0.99) can be achieved; but also shows the limitations as a lower F1 score (0.84) results from sentences that could not be parsed. Our future work will comprise a more comprehensive evaluation of our approach, as well as seeking a solution to the problem of unparsed sentences.

References

Wendy Wibber Chapman, Will Bridewell, Paul Hanbury, F. Gregory Cooper, and G. Bruce Buchanan. 2001. A simple algorithm for identifying negated findings and diseases in discharge summaries. *Journal of Biomedical Informatics*, 34(5):301–310.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A Sag. 2005. Minimal recursion semantics: An introduction. *Research on language and computation*, 3(2):281–332.

Ann A Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage english grammar using hpsg. In *LREC*, pages 591–600. Athens, Greece.

Peter L Elkin, Steven H Brown, Brent A Bauer, Casey S Husser, William Carruth, Larry R Bergstrom, and Dietlind L Wahner-Roedler. 2005. A controlled trial of automated classification of negation from clinical notes. *BMC medical informatics and decision making*, 5(1):1–7.

Yang Huang and Henry J Lowe. 2007. A novel hybrid approach to automated negation detection in clinical radiology reports. *Journal of the American medical informatics association*, 14(3):304–311.

A Johnson, T Pollard, R Mark, S Berkowitz, and S Horng. 2019. Mimic-cxr database.

Tobias Kuhn. 2014. A survey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.

Woodley Packard. 2013. Ace, the answer constraint engine. *URL http://sweaglesw. org/linguistics/ace*.

Luke T Slater, William Bradlow, Dino FA Motti, Robert Hoehndorf, Simon Ball, and Georgios V Gkoutos. 2021. A fast, accurate, and generalisable heuristic-based negation detection algorithm for clinical text. *Computers in biology and medicine*, 130:104216.

Luca Soldaini and Nazli Goharian. 2016. Quickumls: A fast, unsupervised approach for medical concept extraction. In *Medical Information Retrieval (MedIR) Workshop, SIGIR*, pages 1–4, Pisa, Italy.

Yanshan Wang, Liwei Wang, Majid Rastegar-Mojarad, Sungrim Moon, Feichen Shen, Naveed Afzal, Sijia Liu, Yuqun Zeng, Saeed Mehrabi, Sunghwan Sohn, and Hongfang Liu. 2018. Clinical information extraction applications: A literature review. *Journal of Biomedical Informatics*, 77:34–49.

Multi-Phase Context Vectors for Generating Feedback for Natural-Language Based Programming

Michael S. Hsiao
Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA USA 24061

Abstract

Automatic generation of feedback messages in a natural-language based programming for video games is presented. The input sentences are processed in four stages. During each stage, context vectors are aggregated and any violation to a syntactic or semantic rule is reported to allow users to debug and fix the text. The results discuss a list of common errors detected by the proposed method.

1 Introduction

Programming in the user's native language attempts to directly convert instructional text to an executable program. The benefits of such a programming system are many, including increased productivity, reduced effort to learn conventional programming languages and debugging, etc. Thus, proficiency in a NL-based platform will help carry over to learning a conventional object-oriented programming language later on. However, programming in a NL faces many hurdles, including the resolution of ambiguity/imprecision, handling of incomplete sentences, and propagation of context from one sentence to the next. Rather than targeting general-purpose programming with NL, aiming for domain-specific applications should be the first goal. With a specific domain, we can narrow the scope for that target application, and the accepted language resembles somewhat to a controlled natural language (CNL), with a finite set of nouns, verbs, and phrasal structures. However, the grammatical rules used in this work are not as constrained as in most CNLs. Instead, the sentences do not need to conform to rigid grammatical structures.

Motivating Example: Alice wishes to write a program involving a rabbit, fox, and carrots. She writes: *"There are 3 foxes, 20 carrots, and a rabbit. The rabbit moves around. When a rabbit touches a carrot, it eats the carrot. When the rabbit sees a fox, it chases it."*

Such a programming paradigm is much more natural to those who have little experience writing a program, and the users can play the resulting game, providing a positive feedback. Moreover, fixing errors in NL offers an early introduction to debugging. For example, consider the last sentence in the above example: "*When the rabbit sees a fox, it chases it.*" There are multiple possible interpretations for the phrase "it chases it", and the system should be able to offer feedback to the user about this potential bug.

A platform has been constructed for this purpose to create video games. The user enters the program that describes the logic of the game in English. The text is then translated to an executable, playable game via a 4-stage compilation process: syntactic processing, phrasal semantic processing, sentential semantic processing, and code generation. At each stage, a context vector is produced and aggregated. Analyses of the context vectors against syntactic and semantic rules help to generate error messages that pinpoint any imprecise, ambiguous, and/or incorrect expressions. The user can then use these error messages and suggestions to fine-tune and debug their NL text. Analysis of the games written by middle-school students show a list of common errors captured by the system.

The rest of the paper is organized as follows. Section 2 provides the preliminaries and background. Section 3 details the methodology for generating the context vectors and error messages based on these vectors. Section 4 discusses the results and Section 5 concludes the paper.

2 Preliminaries

Let **W** be the sequence of words, w_0, …, w_n, in a sentence; each word in a valid sentence should be able to be mapped to a valid token during first step of parsing. The categories for any valid token is ***L*** *= (E, A, T, P, S)*, where E: the set of entities (or objects), A: the set of actions, T: the set of attributes, P: the set of predicates, and finally, S: the set of optional selectors. Note that all these sets in ***L*** can grow and evolve with time.

For the domain of video games, the set of entities, *E*, is the set of characters involved in the game, such as foxes, rabbits, etc. The set of actions, *A*, may include chase, flee, wander, jump, die, etc. Third, the set of attributes, *T*, includes the color, speed, etc. associated with the characters. Note that the user can add more attributes on the fly. Next, the set of predicates, *P*, may include see(), reach(), touch(), catch(), etc. Finally, the set of selectors, S, allows the user to say something like "when 35 rabbits are gone".

Note that new terms can be learned in *T*. For example, the sentence "*When a fox sees a rabbit, it becomes happy. When a fox is happy, it …*" The term 'happy' is learned and associated with the behavior at run-time, as explained in Hsiao (2018).

In Hsiao (2018), error reporting was limited. Later, in Zhan & Hsiao (2019), an attempt to use machine learning to categorize types of errors was made, again with only limited success. Notably, a small change in a sentence may result in completely different type of error. Thus, accurately mapping an erroneous sentence to a specific error (among a potentially large number of errors) via machine learning alone is likely infeasible. Instead, rules can more accurately capture the formal relations in the context of a sentence that imply an error. In other words, analyses on the aggregated context vector against a rule set can generate the needed error message(s) accurately.

3 Methodology

The four stages for the compilation process is illustrated in Figure 1. The key to our approach is that each stage works on a distinct level of abstract representation of the input text. Context vectors were custom designed for the game domain. However, the context vectors can be tailored according to the needs of a domain. Example fields of the context will be described within each stage.

Stage 1: Syntactic Analysis

Given the sequence of *n* words, w_0, …, w_n, this stage aims to produce a sequence of *m* tokens, **T**, t_0, …, t_m, where $m \leq n$, and a syntactic context vector, **SC**. Every token takes a type as defined in ***L*** explained earlier, such as character, verb, predicate, attribute, etc. Any typo (no match to any word in the lexicon) or grammatical error (such as 'fox chase' instead of 'fox chases') will be output to the user during this stage as well.

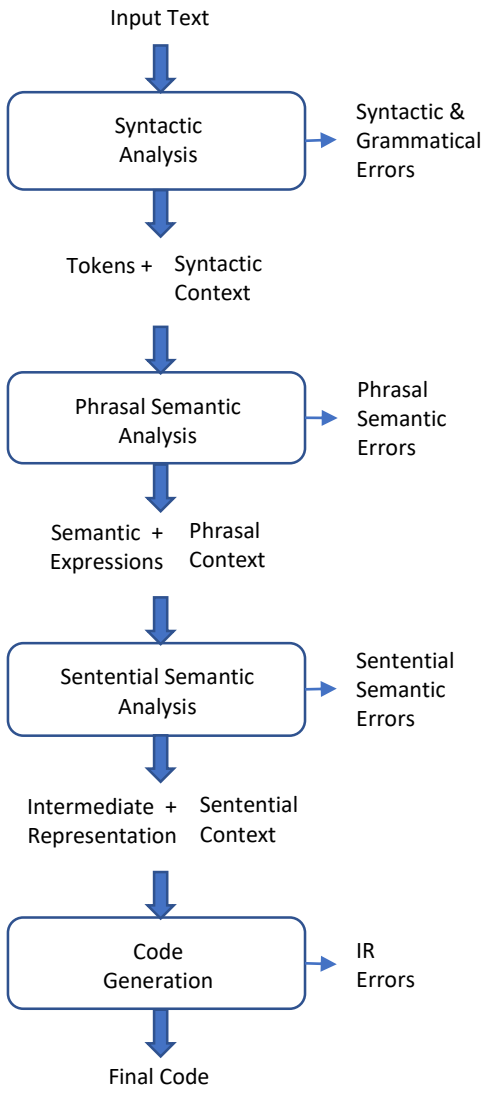


Figure 1: 4-Stage Compilation Process.

Resolution of pronouns is also performed in Stage 1, that binds the pronouns to the corresponding character. Moreover, it learns new words, such as 'happy' as discussed in the preceding section. Such words are not included in the original lexicon, and do not need to be real words. For example, the attribute could be 'xyz' as well. These newly learned words will be

represented as variable attributes in the final code generation. We note also that the syntactic context, **SC**, generated for each sentence is also used to process the next sentence. This is because there might exist relations between consecutive sentences, with words such as "Otherwise" or if the sentence starts with a pronoun, etc.

The **SC** includes the tokens themselves and the statistics of the tokens such as the number of objects, the number and type of verbs, adjectives, colors, adverbs, numbers, etc. Some sentences may contain imprecise verbs, such as 'get' in "*When the fox gets the rabbit, …*" Likewise, there might be usage of other verbs that do not mean their conventional semantics. These instances are also recorded in **SC**. Finally, conjunctives such as 'and' and 'or' are also recorded. The syntactic context, **SC**, along with the token stream, are passed to the next stage to generate the phrasal semantics.

Stage 2: Phrasal Semantic Analysis

In Stage 2, the generated tokens, **T**, and syntactic context, **SC**, from Stage 1 are used to generate semantic expressions, **SE**, and the corresponding phrasal context, **PC**. Consider a simple stream of tokens t_0, t_1, t_2: <obj, fox> <verb, chase> <obj, rabbit> for the text-phrase "*fox chases rabbit*". With frame semantics, the above token stream can be readily converted into a semantic expression of "verb(obj, obj)." Likewise, the phrase "*rabbit is chased by the fox*" maps to the semantic expression "verb(obj, obj)." However, the object identifiers should correctly correspond to the matching actor and actee.

Consider another more sophisticated phrase of "*the happy fox eats the rabbit that is not yellow*", the token stream, **T**, for this phrase produced from Stage 1 is t_i, … t_{i+4}: <adj, happy> <obj, fox> <verb, eat> <adj, not yellow> <obj, rabbit>. Note that the final modifier "*that is not yellow*" is converted to a token <adj, not yellow> in Stage 1 and placed as an adjective modifier *before* the final object. With this token stream, the semantic expression is "verb((adj, obj), (adj, obj))."

This stage also handles conjunctions. For phrases such as "*The foxes and tigers chase the rabbit*," two semantic expressions are generated internally, namely for the phrases "*the foxes chase the rabbit*" and "*the tigers chase the rabbit*".

Any error encountered in the process is also reported. Consider the phrase "*fox chases flees the rabbit*". The tokens would have been <obj, fox> <verb, chase> <verb, flee> <obj, rabbit>. In this case, consecutive verb-tokens are detected, and an error is reported for violating the rule of consecutive action verbs. This rule can be succinctly represented as $t_i \in A \rightarrow t_{i+1} \notin A$, where A is the set of action tokens. Essentially, this rule states that if the i^{th} token is an action, the next token must not be an action token.

Consider another erroneous example, "*fox chases happy*", the dangling adjective, 'happy', without any binding object is a violation and is reported to the user. Verb-tokens such as 'chase' require two objects around it. In this error, there was only one object, fox, which is insufficient to properly form the semantic expression.

Finally, when a conjunctive is about verbs, such as "*the foxes chase and eat the rabbits,*" the system will generate an error message noting the user that such sequences of actions should be split up into different sentences, and provide potential fixes such as "*The foxes chase the rabbits. When a fox catches a rabbit, it eats the rabbit.*" The above error violates the rule that prevents conjunction of action verbs "t_i and t_{i+1}", where both t_i , $t_{i+1} \in A$. The set of semantic rules is manually designed based on the valid phrases allowed in the system. Adding new rules to this set is straightforward.

In addition to generating errors and semantic expressions in Stage 2, a corresponding phrasal context, **PC**, is also produced. The **PC** here refers to the set of semantic expressions (**SE**) discussed above, together with the number and types of semantic expressions, etc. For example, "fox sees rabbit" is a *relational* expression, while "fox chases rabbit" is an *action* expression. Other types of expressions include property expressions, such as "rabbit is happy", and variable expressions, such as "the size of the rabbit equals 3," etc. The **PC**, together with **SE** generated, are passed to Stage 3 to generate the sentential semantics.

Stage 3: Sentential Semantic Analysis

In Stage 3, the goal is to generate the intermediate representation (IR) for each sentence as well as the sentential context, **EC**. Consider the sentence, "*When a fox sees a rabbit, it chases the rabbit*." After Stages 1 and 2, the semantic expressions are **SE**: se_0 = "if see(obj, obj)" and se_1 = "chase(obj, obj)." Each of the <obj> has a unique identifier to bind with the character in question. For this simple example, the IR for the entire sentence

is "if see(obj, obj), then chase(obj, obj)." Consider another simple example with **SE** = "if property(obj, adj)" and "set_color(obj, col)". The resulting IR would be "if property(obj, adj), then set_color(obj, col)."

The types of errors in this stage include the following. Consider the sentence "*When a fox sees a rabbit, it sees the rabbit.*" Here, we have two relational semantic expressions involving predicates without any action expression. Thus, an error message will be produced for violating the missing actionable SE. Here, the violated rule is $\exists se_i \in$ action-SE for every sentence.

Re-writing of the phrases is also performed during this stage for some sentences. For example, if the sentence places the consequent before the antecedent, the system will internally re-write the sentence to preserve canonicity. Finally, resolution of conjunctives such as 'and' and 'or' are performed in this stage as well. Here, the conjunction is analyzed to determine if it is about two separate antecedents or consequents in the sentence. The set of rules for the sentential context is also manually derived, based on the sentences that combine various allowable phrases.

The sentential context, **EC**, for this stage includes the set of IR, together with the type of the IR, as well as the number of antecedents, consequents, complexity of the antecedent, etc. For example, the sentence "*When 35 rabbits are gone, …*" contains a counter 35, along with the IR for the sentence that is given to the subsequent code generation stage.

Stage 4: Code Generation

Finally, with the IR and sentential context, **EC**, Stage 4 generates the output game code based on the **EC**. If there are no errors in any of the previous stages, the IR from Stage 3 would be readily translated to the game code. On the other hand, if there are errors, the context vectors are used to help fill the gap(s) when generating the code. For example, in the first stage, if the number of object tokens is significantly greater than the number of verb tokens (or vice versa), we analyze the token stream further to generate both the code and any additional error message, if appropriate. For instance, in the consequent phrase "*it chases it*", if there are two characters in the antecedent, the system will fill in the two pronouns according to the characters in the antecedent.

We had briefly touched on variables earlier. In addition to Boolean variables such as 'happy', the system also handles non-Boolean variables, such as 'size' in "*When the size of the fox is less than 5, …*" Here, 'size' is a built-in variable available for every object. The user can also define new variables, such as "*When the num_eaten of the rabbit is equal to 5, …*" The variables can also be used in modifier clauses as well. The following sentence is one such example: "*When a rabbit whose size is less than 10 sees a fox, it turns red*." Here, the phrase "*whose size is less than 10*" modifies the rabbit object in the antecedent.

4 Results

The current 4-stage process helps in both the translation of the input text and error-reporting. With this platform (Game Changineer), we are able to produce a wide range of error messages and offer possible fixes to the error. For example, the sentence "*When a rabbit sees a fox, it chases it*" is ambiguous as discussed earlier. The system generates an ambiguity error (noting the two pronouns). Nevertheless, in the presence of such an error, the system will still produce an approximate understanding so that a final game code can still be produced (so that the user can test the game).

Consider another erroneous sentence: "*When a fox sees a rabbit, it chases.*" The verb in the consequent is an action verb (chase), and it is missing a target object. Thus, an error is reported. In addition, the engine tries to remedy the semantic expression by inserting the most suitable missing object from the former semantic expression(s). In this case, it would be the fox chasing the rabbit. This temporary filling of the missing object allows the code generation to complete its generation of the game code. Nevertheless, the above error message is still provided to allow the user to fix the error.

Consider a third error example: "*When the rabbit shoots the fox, the fox runs away*." This sentence may seem correct at first glance, but it is actually ambiguous on the word 'shoot'. Recall that the sentence may be sloppily written by a young user, and as with any NL, the manner in which a verb is used may be imprecise. In this case, there are two possible interpretations of the antecedent clause: (1) '*when the rabbit fires a bullet at the fox*' or (2) '*when a bullet touches the fox*.' With the first interpretation, we know that not every bullet fired will hit the fox. In fact, many

bullets might actually miss the fox. This corresponding error violates the imprecise antecedent verb.

We believe that a good NL-based programming platform should provide a helpful debugging infrastructure to give feedback and guidance to the user on possible misinterpretations. With the error-reporting framework, the system has been piloted at several outreach events to middle school students in the 2020-2021 school year. To the best of our knowledge, no other publicly available system exists that allows users to write video games in English, generates feedback and suggestions on how to fix syntactic and logical errors in the natural language sentences. Because there is no public dataset available, the results are tabulated on anonymized input sentences written by 434 middle school students during the month of March, 2021. Each student created a number of games during the month, and each game may require multiple iterations of debugging. Among the 47,907 errors collected, the 10 most-frequent-occurring errors are reported in Table 1. Both the number of occurrences and type of error are shown.

Table 1: Most frequent-occurring errors

# occur	Error type
742	Spelling error
604	Imprecise verb (such as 'get')
382	Unclear / unsupported phrases
296	Move without direction
294	Incomplete sentence
247	Missing a valid character
245	Ambiguous antecedent
224	Missing a valid verb
223	Imprecise word
178	Logical error on sequencing events

Based on Table 1, it is not surprising that spelling error ranked highest. It is worth noting that the system may generate multiple errors for a given erroneous sentence. For example, a spelling error may also result in a "missing a valid character" or "unclear / unsupported phrase" error. The second most frequent error was the use of imprecise verbs. These occur with phrases such as "*fox gets the rabbit*" or "*the bird is hit*". These phrases can have multiple interpretations, including "touch", "eliminates", or "is shot". Next, unclear / unsupported phrases include those facetious phrases such as "the fox farts". Next, an example of "move without direction" is the phrase "*the rabbit moves at 2 pixels per frame*". This can be interpreted as either "*wanders around at 2 pixels per frame*" or "*the speed of the rabbit is 2*." Without clarity, the system chooses the latter.

An example of an incomplete sentence is "*When the rabbit sees a carrot, chase.*" Here the objects in the consequent are missing and need to be filled. Finally, the logical error on sequencing events is an interesting type of error. For example, a sentence "*When the rabbit dies, the game is over*" is correct in itself, but will result in such an error if there was no earlier description on how the rabbit can die before this sentence. Screen shots of two games created are illustrated in Figure 2. Many more games are available on the website.

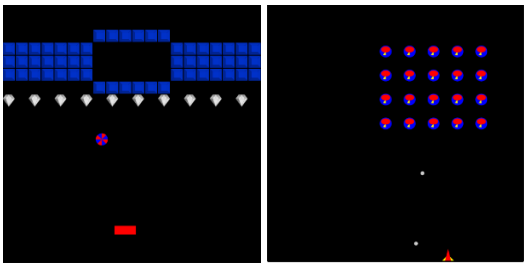


Figure 2: Breakout and Space Invaders Type Games

5 Conclusions

We have presented a 4-stage process to generate error messages for English sentences that could not be processed. At each stage, a context vector is constructed and propagated to the next stage. Analysis of the context vectors plays a critical role in both the generation of game code and any error messages that pinpoint imprecise, incomplete, and/or incorrect expressions. These error messages help guide the user to correct their errors. Results from games created by Middle-school students show the potential of such a framework to help them bring their designs to completion.

References

Game Changineer website: https://gc.ece.vt.edu

Hsiao, M. S. (2018). Automated program synthesis from object-oriented natural language for computer games. Proc. Int. Workshop on Controlled Natural Language.

Zhan. Y., & Hsiao, M. S. (2019). Multi-label classification on natural language sentences for video game design. Proc. IEEE Int. Conf. on Humanized Computing and Communication.