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.

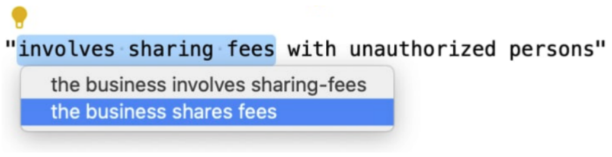
Top-level ambiguity The first step in disambiguation is to temporarily remove all postmodifier adjuncts (see Table 1) from the initial trees, thereby reducing the tree depth until the heads with light modifiers remain. The goal is to sieve out the main structure, and make it easier for the user to disambiguate 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



```
"involves sharing fees with unauthorized persons"
```

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.