Autoformalisation via Grammatical Framework

Shashank Pathak

January 11, 2024

The University of Manchester

Contents

GFLean

GFLean

GFLean[1] is a Haskell program that converts natural language text blocks to Lean 4 terms.

- The input it accepts is written in a Controlled Natural Language (CNL), which we call **Simplified ForTheL**.
- The output it produces needs some post-processing before it can be passed to Lean.
- The parsing and linearisation is done via a tool called **Grammatical Framework** (GF)[2].
- GFLean only converts statements and not proofs.

[1] https://github.com/pathakshashankgit/GFLeanTransfer
[2] https://www.grammaticalframework.org/

Translation Examples

```
ex. assume x is a rational number . assume x is equal to
( 2 * 2 ) . then x is greater than 3 .
```

```
example ( x : ℚ ) ( h? : x = ( 2 * 2 ) ) : x > 3 := sorry
```

```
ex. every odd integer greater than 2 is greater than 1 .
```

```
example : ∀ ( ( x 6 ) : ℤ ), ( ( odd ( x 6 ) ∧ ( x 6 ) > 2 ) →
    ( x 6 ) > 1 ) := sorry
```

Translation Examples (cont'd)

```
ex. assume y is an integer . assume for no even integer x ,
x is odd . then y is not odd .

example ( y : ℤ ) ( h? : ∀ ( x : ℤ ), ( even x → ( ¬ odd x ) )
    ): ( ¬ odd y ) := sorry
```

```
ex. assume x is a real number . assume ( 2 + 2 ) is less
than ( -1 ^ 3 ) . then no nonnegative integer a such that
a is positive is not even .

example ( x : ℝ ) ( h? : ( 2 + 2 ) < ( -1 ^ 3 ) ) : ∀ ( a : ℤ ),
( ( nneg a ∧ pos a ) → ( ¬ ( ¬ even a ) ) ) := sorry
```

Grammatical Framework

Grammatical Framework

GFLean uses **Grammatical Framework** (GF)[3] for parsing the input and linearizing the output.

GF is a functional programming language specifically designed to write and implement grammars.

A GF program is a GF grammar and each grammar has one **abstract syntax** (AS) and multiple corresponding **concerete syntaxes** (CSs).

CSs are supposed to encode the language specific details like genders and cases, and AS is supposed to encode the meaning.

The AS acts as a bridge between the various CSs.

[3]https://www.grammaticalframework.org/

Grammatical Framework

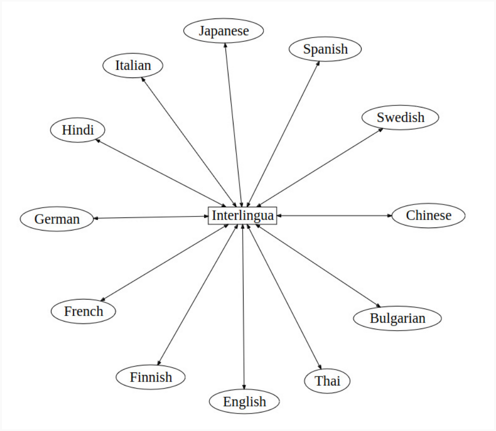


Figure 1: Using abstract syntax as interlingua[RAGK20].

Grammar writing in GF

GF is a high-level language for writing grammars: **write the grammar rules, get the parser for free**.

One can use **records, tables** and **parameters in concrete syntaxes** to model language-specific agreements like genders, numbers and cases.

GF already has natural language grammars defined, called the **Resource Grammar Library** (RGL), which can be imported like software libraries.

As of 2019, RGL has implementations of 35 natural languages with a common abstract syntax[RAGK20].

GF grammars can be embedded in Haskell programs, and the abstract syntax trees can be manipulated as Haskell objects.

GF makes the task of grammar writing **modular** by distributing functionalities between the abstract and the concrete syntax.

Simplified ForTheL

Simplified ForTheL

The input GFLean accepts is written in a controlled natural language (CNL) called **Simplified ForTheL**.

Simplified ForTheL is a simplified version of the CNL **ForTheL**[4] on which the System for Automated Deduction (SAD)[LVDP02] and the Naproche CNL[DLKL+21] is based.

Our implementation of Simplified ForTheL in GF is based upon a ForTheL implementation in GF[SAK20].

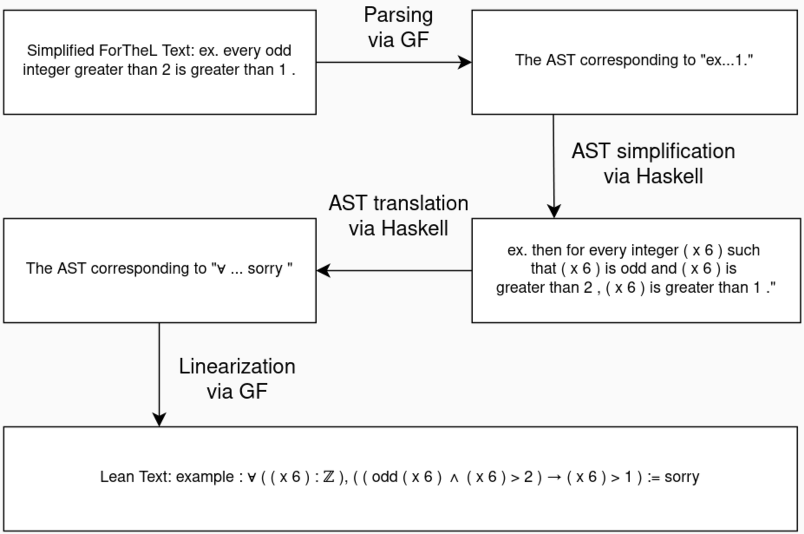
[4]http://nevidal.org/download/forthel.pdf

ForTheL vs Simplified ForTheL

ForTheL	**Simplified ForTheL**
Any number of left adjectives: *x is an odd nonnegative … prime integer.*	Just one left adjective: *x is an odd integer. x is nonnegative. x is prime.*
Conjunction of predicate list: *x is odd and greater than* 4.	A single predicate: *x is odd and x is greater than* 4.
Conjunction of term list: *x and y are odd.*	A single term: *x is odd and y is odd.*
Dynamic (can be extended during runtime).	Static (can't be extended during runtime).
Macro-level grammar is geared towards SAD[LVDP02].	Macro-level grammar is geared towards Lean.

The workings of GFLean

The GFLean Pipeline


Simplified ForTheL Text: ex. every odd integer greater than 2 is greater than 1 .
Parsing via GF
The AST corresponding to "ex...1."
AST simplification via Haskell
ex. then for every integer (x 6) such that (x 6) is odd and (x 6) is greater than 2 , (x 6) is greater than 1 ."
AST translation via Haskell
The AST corresponding to "∀ ... sorry "
Linearization via GF
Lean Text: example : ∀ ((x 6) : ℤ), ((odd (x 6) ∧ (x 6) > 2) → (x 6) > 1) := sorry

Parsing Simplified ForTheL Expressions

We use GF for parsing the Simplified ForTheL expression.

We wrote an abstract syntax and a concrete syntax for the Simplified ForTheL grammar.

The concrete syntax is still crude, and it can be made better to not accept ungrammatical sentences like:

```
assume x are an odd integers .
```

AST Simplifications

GF grammars can be embedded into Haskell programs and the abstract syntax trees (ASTs) can be manipulated as Haskell objects.

We do some simplifications on the ASTs which result in the following:

- every entity gets a name,
- variable names are unified, and
- the sentences are written in a specific form which eases the translation process.

For example, the AST for

```
ex. every odd integer greater than 2 is greater than 1 .
```

becomes the AST for

```
ex. then for every integer ( x 6 ) such that ( x 6 )
is odd and ( x 6 ) is greater than 2 , ( x 6 )
is greater than 1 .
```

AST Translations

The simplified ASTs are translated into ASTs for the Lean expressions.

How the natural language quantifiers like *every*, *some*, *no* and *negation* interact is implemented in this step.

Thus, the AST for

```
ex. then for every integer ( x 6 ) such that ( x 6 )
is odd and ( x 6 ) is greater than 2 , ( x 6 )
is greater than 1 .
```

becomes the AST for

```
example : ∀ ( ( x 6 ) : ℤ ), ( ( odd ( x 6 ) ∧ ( x 6 ) > 2 )
→ ( x 6 ) > 1 ) := sorry
```

Linearising as Lean Expressions

Once again, we use GF to linearize the translated ASTs to Lean expressions.

The output produced by GFLean **should be post-processed** for it to type-check as a correct Lean expression.

For example, variables which are not present in the input are written as

```
( x 1 ), ( x 2 )
```

etc. and hypothesis variables are written as

```
h?
```

Limitations and Further Plans

Limitations

Sentences where quantified notions are in the predicate can not be simplified and translated by GFLean.

```
Every integer is greater than some real number.
```

Sentences of the form “x is an *adjective common noun*.” are simplified but not translated by GFLean.

```
assume x is an even integer greater than 32 .
```

gets simplified to

```
assume x is a integer x such that x is even and x is
greater than 32 .
```

Limitations (Cont'd)

- Simplified ForTheL is too simple to be used as a CNL for mathematics.
- Not possible to extend the grammar during runtime.
- Using '(,)' in order to disambiguate symbolic expressions.
- Tiny lexicon.

Further Plans

- Support for quantified notions in the predicate (by understanding how scoping works in the language of mathematics).
- Support for sentences of the form "x is an *adjective common noun*".
- Extending GFLean to ForTheL.
- Implementing operator and logical precedence in the natural language concrete syntax.
- Increasing the lexicon.

Where is Lean in the loop?

Nowhere.

In order to have dynamicity and communication with Lean, the whole of GFLean should be implemented in Lean itself.

Lean 4 is a full-fledged programming language, with a good metaprogramming support.

How?

Why?

Bibliography i

Adrian De Lon, Peter Koepke, Anton Lorenzen, Adrian Marti, Marcel Schütz, and Makarius Wenzel, *The isabelle/naproche natural language proof assistant*, Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28, Springer International Publishing, 2021, pp. 614–624.

Alexander Lyaletski, Konstantine Verchinine, Anatoli Degtyarev, and Andrey Paskevich, *System for automated deduction (sad): Linguistic and deductive peculiarities*, Intelligent Information Systems 2002: Proceedings of the IIS'2002 Symposium, Sopot, Poland, June 3–6, 2002, Springer, 2002, pp. 413–422.

Bibliography ii

- Aarne Ranta, Krasimir Angelov, Normunds Gruzitis, and Prasanth Kolachina, *Abstract syntax as interlingua: Scaling up the grammatical framework from controlled languages to robust pipelines*, Computational Linguistics **46** (2020), no. 2, 425–486.
- Jan Frederik Schaefer, Kai Amann, and Michael Kohlhase, *Prototyping controlled mathematical languages in jupyter notebooks*, Mathematical Software–ICMS 2020: 7th International Conference, Braunschweig, Germany, July 13–16, 2020, Proceedings 7, Springer, 2020, pp. 406–415.

Thank you!