GFLean: Autoformalisation for Lean via GF

HIM Trimester Program, Prospects of Formal Mathematics

Shashank Pathak

July 22, 2024

MANCHESTER 1824
The University of Manchester

Contents

Formalisation and Autoformalisation

Formalisation

Formalisation: Converting a text from a natural language to a formal language.

Formalisation of mathematical text is a complex task.

Perform inferences, fill gaps, reformulate arguments

Autoformalisation Puzzles

- Every continuous map from a disk onto *itself* has a fixed point.
  Itself - continuous map or disk ?

- The natural number p is prime.
  The natural number p is greater than 1.

- Then $n \in \{1, 2\}$.
  For $n \in \{1, 2\}$, we have $n > 0$.

- The sets A and B are *empty*.
  The sets A and B are *disjoint*.

- The addition of residue classes $\mathbb{Z}/n\mathbb{Z}$ is associative.

Autoformalisation Puzzles (Cont'd)

- Show that a group of order 5 *must be* abelian.
  Every integer greater than 1 *can* be represented uniquely as a product of prime numbers, up to the order of the factors.
- For all odd n, we have $8|n^2 - 1$.
- Every continuous map from a disk into itself has a fixed point.
- A relation between A and B is a subset of $A \times B$.

An ideal autoformalisation program should be able to do all these.

It is worthwhile to study how we write and understand mathematics.

GFLean

Language Translation ≈ Code Compilation

Compilation: Translation from one formal language to another.

Our method: Translation from a formally-defined subset (aka controlled natural language) of the language of mathematics to a formal language.

Input: Looks natural, acts formal.

Output:

GFLean

GFLean[1] - Haskell program which converts natural language text blocks to Lean terms.

- Input in a controlled natural language (CNL), which we call Simplified ForTheL.
- Parsing and linearisation done via **Grammatical Framework** (GF)[2].

Only converts statements but not proofs.

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

Translation Examples

Ex. Assume x is a rational number. Assume x is equal to 2 + 2 * 2. Then x is greater than 3.

```
example (x : ℚ) (h39 : x = (2 + (2 * 2))) : x > 3 := sorry
```

Ex. Assume y is an integer and for no positive integer x, y is greater than x. Then y is less than or equal to 1.

```
example (y : ℤ) (h52 : ∀ (x : ℤ), (pos x → (¬ y > x))) :
y ≤ 1 := sorry
```

Translation Examples (cont'd)

Ex. Assume n is an odd integer greater than 1 and x is a rational number less than 0. Then every real number greater than n is greater than x.

```
example (n : ℤ) (h88 : odd n) (h75 : n > 1) (x : ℚ)
(h54 : x < 0) : ∀ (x51 : ℝ), (x51 > n → x51 > x) := sorry
```

Ex. Assume x is an even integer greater than 32. Then x is greater than every integer less than 32.

```
example (x : ℤ) (h70 : even x) (h56 : x > 32) : ∀ (x34 : ℤ),
(x34 < 32 → x > x34) := sorry
```

GFLean Execution Steps

GFLean uses

1. **Grammatical Framework** (GF) for parsing the input and producing abstract syntax trees (ASTs).
2. Haskell for AST manipulations.
3. GF again to produce Lean expressions from the manipulated ASTs.

Grammatical Framework

Grammatical Framework

GF: Functional programming language specifically designed to implement grammars.

Easy to build natural language translation programs using GF.

Abstract and Concrete Syntaxes

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

CSs encode the language specific details like genders and cases, and AS encode the meaning.

The AS acts as a bridge between the various CSs.

Abstract and Concrete Syntaxes (Cont'd)

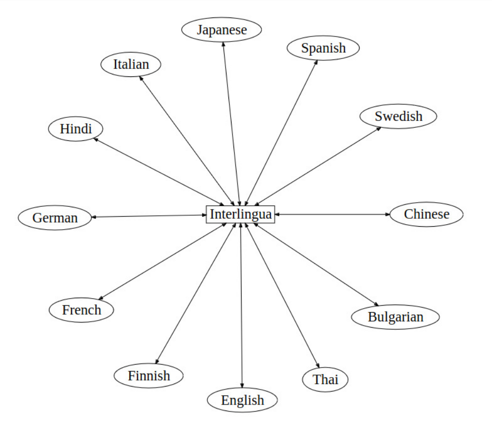


Figure 1: Using abstract syntax as interlingua[3].

[3] Ranta, Aarne, et al. "Abstract syntax as interlingua: Scaling up the grammatical framework from controlled languages to robust pipelines." Computational Linguistics 46.2 (2020): 425-486.

Grammar writing in GF

GF : High-level language for writing grammars.

Write the grammar rules, get the tokenizer, parser and type-checker for free.

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

GF grammars are modular: Distributed functionalities between the AS and the CSs.

Resource Grammar Library (RGL): Natural language grammar library in GF. RGL has natural language grammars for 35 languages with the same AS.

GF Grammars can be embedded in Haskell and the abstract syntax trees can be manipulated as Haskell objects.

A Small GF Grammar

```
abstract Demo = {
...
Nzero, Greater1 : Var -> Prop;
Imp : Prop -> Prop -> Prop;
... }
```

```
concrete DemoEng of Demo = {
...
Nzero var = var ++ "is nonzero";
Greater1 var = var ++ "is greater than 1";
Imp prop1 prop2 = "If" ++ prop1 ++ ", then"
... }
```

```
concrete DemoMath of Demo = {
...
Nzero var = "~ ("++ var ++ " = 0)";
Greater1 var = var ++ " > 1";
Imp prop1 prop2 = "(" ++ prop1 ++ "→" ++ prop2 ++ ")";
... }
```

A small GF Grammar (cont'd)

```
Demo> parse -lang=Eng "For each x1 , There exists an x2 such
that If x1 is nonzero , then x2 is greater than 1 "
| linearise -lang=Math
```

produces the following output

$(\forall\ x1\ ,\ (\exists\ x2\ ,\ (\ \tilde{}\ (\ x1 = 0) \rightarrow x2 > 1\ )\ )\ )$

Abstract Syntax Trees (ASTs)

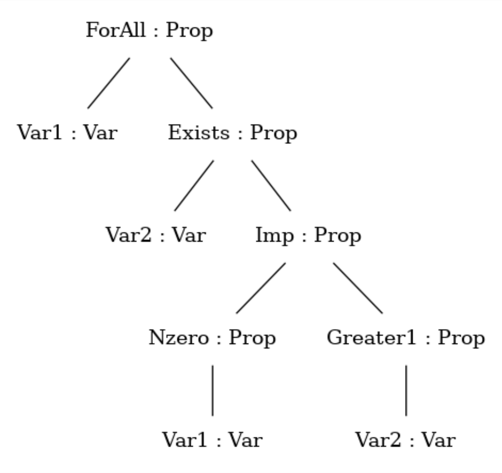


Figure 2: The AST produced by GF after parsing "`For each x1 , There exists an x2 such that If x1 is nonzero , then x2 is greater than 1.`"

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) and the Naproche CNL is based.

Our implementation of Simplified ForTheL in GF is based upon a ForTheL implementation in GF by Schaefer.

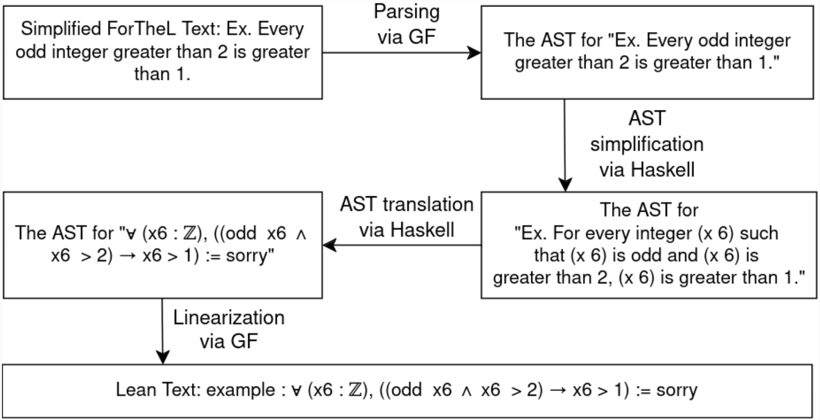
[4] http://nevidal.org/download/forthel.pdf

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.	Macro-level grammar is geared towards Lean.

The workings of GFLean

The GFLean Pipeline



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.

Implemented operator precedence and logical precedence by using record and tables in the concrete syntax.

`2 + 2 * 2` is `2 + (2 * 2)`

`A and B iff C` is `(A and B) iff C`

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

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

- every unnamed 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. Assume x is an integer greater than 2. Then no odd
integer less than 1 is greater than x.
```

becomes the AST for

```
Ex. Assume x is an integer. Assume x is greater than 2.
Then for no integer (x 29) such that (x 29) is odd and
(x 29) is less than 1, (x 29) is greater than x.
```

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, after Translation, the AST for

```
Ex. Assume x is an integer. Assume x is greater than 2.
Then for no integer (x 29) such that (x 29) is odd and
(x 29) is less than 1, (x 29) is greater than x.
```

becomes the AST for

```
example (x : ℤ) (h70 : x > 2) : ∀ (x29 : ℤ), ((odd x29 ∧
x29 < 1) → (¬ x29 > x)) := sorry
```

Linearising as Lean Expressions

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

Pre-processing: Converting all text to lowercase.

Post-processing: Deleting extra whitespaces and giving each variable hypothesis a unique name.

Evaluation, Limitations and Further Plans

Evaluation

Can formalise statements from a chapter of an introduction to proofs textbook[5].

42 out of 62 statements but with some rephrasing.

[5] Chartrand, G., Polimeni, A.D., Zhang, P.: Mathematical proofs. Chap. 3, pp. 81–104. Pearson (2017)

Evaluation (Cont'd)

[Exercise 3.10] If a and c are odd integers, then $ab + ac$ is even for every integer b.

```
Ex. Assume a is an odd integer and c is an odd integer.
Then for every integer b, a * b + a * c is even.
```

```
example (a : ℤ) (h78 : odd a) (c : ℤ) (h57 : odd c) :
∀ (b : ℤ), even ((a * b) + (a * c)) := sorry
```

Evaluation (Cont'd)

[Theorem 3.17] Let a and b be integers. Then ab is even if and only if a is even or b is even.

```
Ex. Assume a is an integer and b is an integer. Then a * b
is even iff a is even or b is even.
```

```
example (a : ℤ) (b : ℤ) : (even (a * b) ↔ (even a ∨ even b)) :=
    sorry
```

Evaluation (Cont'd)

[Exercise 3.1] Let $x \in \mathbf{R}$. If $0 < x < 1$, then $x^2 - 2x + 2 \neq 0$.

```
Ex. Assume x is a real number. Assume x is greater than 0
and x is less than 1. Then x ^ 2 - 2 * x + 2 is not
equal to 0.
```

```
example (x : ℝ) (h64 : x > 0) (h51 : x < 1) : (((x ^ 2) - (2 *
    x)) + 2) ≠ 0 := sorry
example (x : ℝ) (h68 : x > 0) (h55 : x < 1) : (¬ (((x ^ 2) - (2
    * x)) + 2) = 0) := sorry
```

Evaluation (Cont'd)

[Exercise 3.37] Let $x, y, z \in \mathbf{Z}$. If exactly two of the three integers x, y, z are even, then $3x + 5y + 7z$ is odd.

```
Ex. Assume x is an integer, y is an integer and z is an
integer. Assume x is even, y is even and z is not even or
x is even, y is not even and z is even or x is not even,
y is even and z is even. Then 3 * x + 5 * y + 7 * z
is odd.
```

```
example (x : Z) (y : Z) (z : Z) (h158 : ((even x ∧ (even y ∧ (¬
    even z))) ∨ ((even x ∧ ((¬ even y) ∧ even z)) ∨ ((¬ even
    x) ∧ (even y ∧ even z))))) :
odd (((3 * x) + (5 * y)) + (7 * z)) := sorry
```

Limitations

Accepts a small fragment of the language of mathematics.

- Simplified ForTheL is too simple.
- Lexicon is tiny.

Can accept ungrammatical sentences like

```
Assume x are an odd integers.
```

Not possible to extend the grammar during runtime.

Summary and Further Work

GF is a useful tool to build modular and potentially scalable rule-based autoformalisation programs.

We have used GF to produce GFLean.

GFLean correctly formalises 42 out of 62 statements from a textbook.

Next:

- Using records, tables and parameters to make concrete syntaxes better.
- Extend GFLean's input language.
- Use Lean to ease some parts of the process?

More information[6].

[6] Pathak, Shashank. "GFLean: An Autoformalisation Framework for Lean via GF." arXiv preprint arXiv:2404.01234 (2024).

Thank you! Questions?