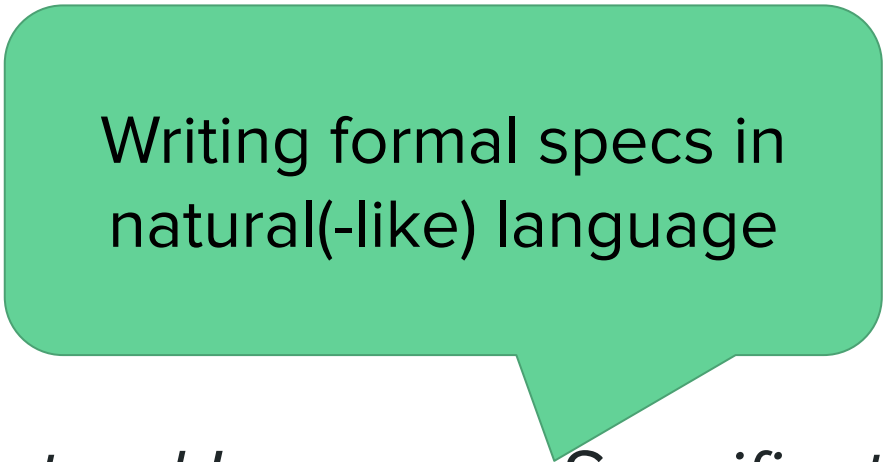


Trustworthy Formal Natural Language Specifications

Colin S. Gordon and Sergey Matskevich
Onward! 2023, Cascais, Portugal

Trustworthy Formal Natural Language Specifications



Writing formal specs in
natural(-like) language

Trustworthy Formal *Natural Language Specifications*

Trustworthy **Formal Natural Language** Specifications



Using formalized
fragments of natural
language

Writing formal specs in
natural(-like) language

Trustworthy **Formal *Natural Language*** Specifications

Using formalized
fragments of natural
language

Mechanized Proofs of Software Correctness

- Very powerful, now widely explored, with major successes
 - CompCert, seL4, CertiKOS, ...
- Serious limitation: What if you formalize the wrong spec?
- Arises from the *specification gap*: most descriptions start out in natural language, and are translated *by (fallible) humans* to formal claims
 - Less technical users have to trust this was done correctly
 - At best, (careful) notes about translation, but rare in practice
 - Not easily auditable
- Ideally:
 - Automatically checked translation
 - With evidence (think LCF-style provers)

Goals

- Write specifications in [formal (natural) language]
- Ease writing of some specifications
- Extend verification of properties up through a form *less technical users can understand*
- Modular extensibility
- Non-goal: *replacing* formal specifications entirely

Writing Formal Specs in Natural Language

- Not a new idea!
 - Lots of work going back to the 70s, heavy in the 80s, more with the advent of LLMs
 - English for first-order logic (ACE, ...)
 - Pseudo-English programming (COBOL, Applescript, ...)
- So what will we do differently?
 - Integrated into a proof assistant (Lean 4), *as a library*
 - No LLMs
 - We will *supplement*, not supplant, advanced interfaces
 - We will embrace, not ignore, theoretical and empirical linguistics

Categorial Grammars

- Large family of techniques rooted in:
 - Theoretical, formal linguistics
 - Substructural logic
- Long history (Ajdukiewicz, Bar-Hillel, Lambek, van Benthem, ...)
 - Treatments of English, German, Hindi, Japanese, Arabic, French, ...
- Not totally unknown to PL people! (Kiselyov, Kokke, Shan, ...)
 - That work was focused on linguistics outcomes; we're building verification tools
- Give grammatical types to words (lexical entries)
- Grammatical types + logical rules guide fragment combinations
 - New grammar coverage is often just a new rule or lexical entry (from existing linguistics work)
- Parsing is now proof search!
- *Semantic* parsing: each rule combines semantics of the parts into semantics of the whole
 - The principle of *compositionality* in linguistics; aka, denotational semantics

Categorial Grammars: Basics

Rules, e.g.

$$\frac{\Gamma \vdash B/A \Rightarrow f \quad \Delta \vdash A \Rightarrow a}{\Gamma \Delta \vdash B \Rightarrow (f a)}$$

$$\frac{\Gamma \vdash A \Rightarrow a \quad \Delta \vdash A \backslash B \Rightarrow f}{\Gamma \Delta \vdash B \Rightarrow (f a)}$$

Where if $\Gamma \vdash C \Rightarrow e$ then C determines e 's type — $e : \text{LCJ}$

Plus: other rules can push beyond CFLs to CSLs and Turing-Complete langs

$$\frac{\text{three} \vdash \text{NP} \Rightarrow 3 \quad \frac{\text{is} \vdash (\text{NP} \backslash \text{S}) / \text{ADJ} \Rightarrow \lambda p. \lambda x. (p x) \quad \text{even} \vdash \text{ADJ} \Rightarrow \text{even}}{\text{is even} \vdash \text{NP} \backslash \text{S} \Rightarrow \lambda x. (\text{even } x)}}{\text{three is even} \vdash \text{S} \Rightarrow \text{even } 3}$$

Our prototype builds this structure, which acts as a certificate.

Lean Typeclasses

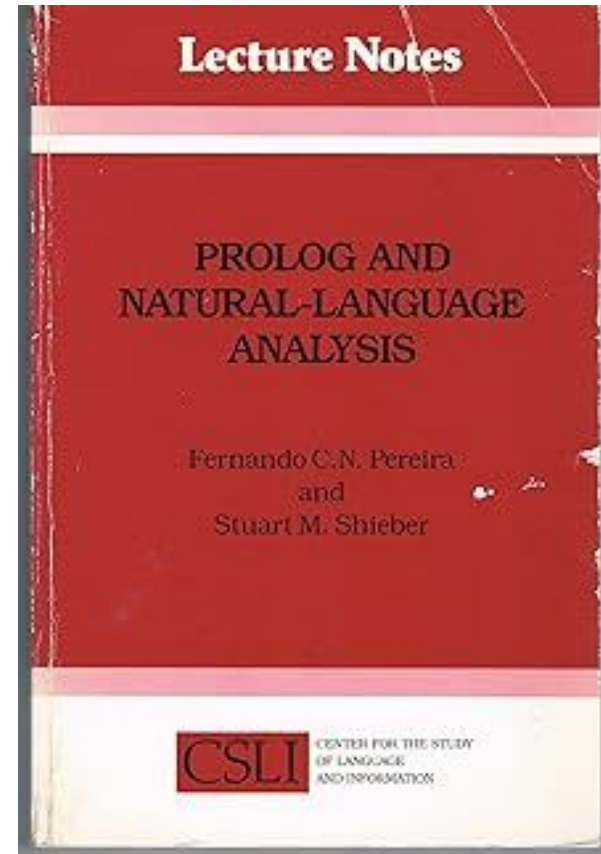
- Recall: typeclasses are a way to get the compiler to provide implicit arguments
 - Draws heavily on logic programming
 - Quite popular in Haskell, Scala
- Core idea:
 - `spec (ws : list string) [syn:Synth ws S] : Prop := syn.sem`
 - `spec ("three"::"is"::"even"::[])` causes Lean to search for a `Synth ("three"::"is"::"even"::[]) S`
 - `[Synth ws c]` corresponds to $ws \vdash c$
 - Semantics are a member of the typeclass
 - Each rule of $\Gamma \vdash C \Rightarrow e$ becomes an *instance* for `Synth`, e.g.
 - `instance rapp {Γ Δ A B} [Synth Γ (B/A)][Synth Δ A] : Synth (Γ++Δ) B` where ...
 - Lean searches possible ways to combine instances to match a goal
- Plus a macro so we're not writing string lists everywhere

Specs in Lean: Examples

- To VS Code!

Making It Fast

- Lean 4 implementation choices
- 1980s Prolog optimizations
 - Difference lists
 - Interning as numbers



Evaluating

- Grabbed some specifications from *Verified Functional Algorithms*
 - Appel's Software Foundations volume
 - Early chapters
- Wrote English as close as possible to the original
- Noted divergence due to prototype limits (grammar coverage), spec ambiguity, interesting linguistic phenomena

Specs

#	Ch	Original (formal) and Generated (if different)
1	Sort	Original: “insertion maintains sortedness” (<code>forall</code> a l, sorted l $\rightarrow$ sorted (insert a l)) Translated (6s): “insertion of any natural maintains sortedness” (α -equivalent)
2		Original: “insertion sort makes a list sorted” (<code>forall</code> l, sorted (sort l)) Translated (5.32s): “sort sorts any list of naturals” (α -equivalent)
3		[No Original English] (<code>forall</code> x l, Permutation (x::l) (insert x l)) Proposed & Translated (7.4s): “insert is a permutation of cons” (α -equivalent)
4		Original & Translated (1.84s): “sort is a permutation” (<code>forall</code> l, Permutation l (sort l))
5		[No original English] (<code>forall</code> l, sorted (a l) $\wedge$ Permutation l (a l)) Proposed & Translated (5.72s): “sort is a sorting permuting algorithm” (<code>exists</code> a, (<code>forall</code> l, sorted (a l)) $\wedge$ (<code>forall</code> l, Permutation l (a l)) $\wedge$ a=sort)
6	Multiset	Original & Translated (1.27s): “union is associative” (<code>forall</code> a b c, union a (union b c)=union (union a b) c)
7		Original & Translated (1.24s): “union is commutative” (<code>forall</code> a b, union a b = union b a)
8		Original: “insert produces the same contents as merely prepending the inserted element to the front of the list” (<code>forall</code> x l, contents (insert x l) = contents (x :: l)) Translated*: “insertion and cons of any value yield equal contents” (α -equivalent)
9		Original & Translated (1.27s): “sort preserves contents” (<code>forall</code> l, contents l = contents sort l)
10		[No original English] (<code>forall</code> l, contents l = contents (sort l) $\wedge$ sorted (sort l)) Proposed & Translated (7.18s): “sort preserves contents and sorts” (α -equivalent)

Table 1. Translation experiments with *Verified Functional Algorithms* specifications.

Doesn't parse on my Mac, parses on a new PhD student's Linux box!

Interesting Linguistics Findings

- One spec we didn't translate, rewritten, would refer to *the syntax of the expression denoting the operations*, rather than operations themselves!
 - “Multisets in a nested union can be swapped”
 - $\text{union } a (\text{union } b \ c) = \text{union } b (\text{union } a \ c)$
 - (We also think this is slightly vague)
- Linguistic semantics have never been formally connected to parametric polymorphism
 - Universe size issues with $L \rightarrow J : \text{Category} \rightarrow \text{Type}$
 - “Every BST” is actually *three* quantifiers: $\forall K:\text{Type}, \forall V:\text{Type}, \forall t:\text{BST}\langle K, V \rangle, \dots$

Wrapping Up

- We can support natural language specifications in current proof assistants
 - With evidence
 - With good (extensible) linguistic coverage
 - As an extensible library!
- Interesting future work
 - Linguistics of how we talk about formulae
 - Universes in the natural language we use
- More details and nuances in the paper!

Backup Slides

What we're *not* doing: LLMs

- Lots of recent attention on autoformalization with LLMs
- Very cool! Open-ended! Requires no special linguistic background!
- If something goes wrong, fix it yourself
 - Cannot modularly correct individual terms
- Want to stop it from happening again?
 - OpenAI's model? You're stuck
 - Your model? Retrain! And hope nothing else changed...
- Want to understand what went wrong?
 - You can get a heat map that marks which words most influenced output
 - No linguistically-grounded explanation