Structure-Aware Neural Representations of Agda Programs

Konstantinos Kogkalidis[1,2][0009−0008−1535−9717]
Orestis Melkonian[3][0000−0003−2182−2698]
Jean-Philippe Bernardy[4,5][0000−0002−8469−5617]

[1] Aalto University
[2] University of Bologna
[3] Input Output (IOG)
[4] University of Gothenburg
[5] Chalmers University of Technology

Abstract. We introduce an ML-ready dataset for dependently-typed program-proofs in the Agda proof assistant, and present the underlying extraction tool. We then demonstrate how a bidirectional Transformer encoder retrofitted with structure-specific adjustments can achieve high performance in a realistic premise selection setup. Unlike most go-to LLM architectures, both the dataset and the neural model work in tandem to maximally preserve the typing information offered by Agda. The resulting pipeline is agnostic and impervious to most syntactic confounds, producing inferences on the basis of type structure alone. Empirical evidence support this being a general, robust, and efficient modeling approach.

Keywords: Premise Selection · Dependent Types · Agda · Representation Learning.

Motivation. The code below suggests how one could go about formalizing the commutative property of addition using the Agda proof assistant.

```
open import Relation.Binary.PropositionalEquality using (_≡_; refl; cong; trans)


data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ

_+_ : ℕ → ℕ → ℕ
zero  + n = n
suc m + n = suc (m + n)


+-comm : (m n : ℕ) → m + n ≡ n + m
+-comm zero    zero    = refl
+-comm zero    (suc n) = cong suc (+-comm zero n)
+-comm (suc m) zero    = cong suc (+-comm m zero)
+-comm (suc m) (suc n) = cong suc (trans (+-suc m n) (+-comm (suc m) n))
  where +-suc : ∀ m n → m + suc n ≡ suc (m + n)
        +-suc zero    n = refl
        +-suc (suc m) n = cong suc (+-suc m n)
```

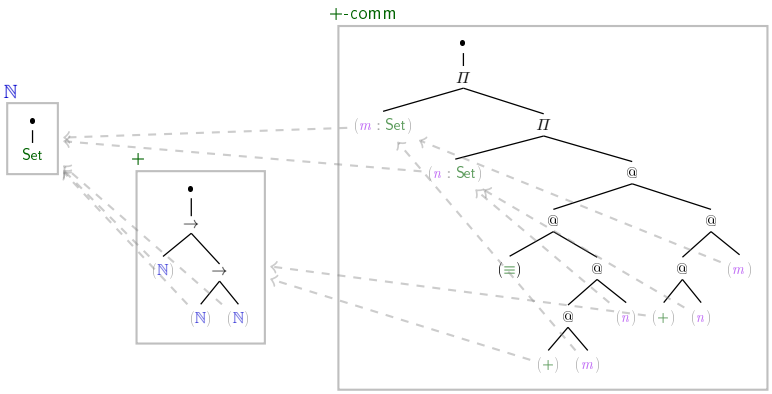
Agda has colored the code for us; its ability to do so is an assertion the code does indeed type check, while the colors assigned help us to visually differentiate substrings according to their syntactic function (*e.g.*, built-ins, data types, named

variables, functions/properties *etc.*). Also worth noting is that the names of the two naturals used in this proof don't really matter, since the two are universally quantified; *m* and *n* might as well have been *x* and *y*, *foo* and *bar*, or *Alice* and *Bob* for that matter. When integrating the proof assistant with a machine learning system, how one treats the code matters. Treating the code as a flat string obfuscates these (and many other) useful hints of syntactic wisdom, demoting the type checker from a valuable collaborator to a deferred sanity filter.

Contributions. Here, we do the code justice. Our contributions are two-fold:

- **Machine Learning for Agda** [2]
  We develop a package to faithfully extract the skeleton structure of dependently-typed program-proofs from type-checked Agda files. We apply the algorithm on Agda's public library ecosystem and release the result as a massive, highly elaborated ATP dataset.
- **Representation Learning for Dependent Types** [3]
  Capitalizing on this new resource, we present a representation learning model for expressions involving dependent types. Contra prior work, the model is structure-faithful, being invariant to α-renaming, superficial syntactic sugaring, scope permutation, irrelevant definitions, *etc.*

Methodology. We use Agda's type-checker to find all possible holes in all written proofs. For each hole, we record the goal type and the typing context. Ground truth corresponds to the subset of the context that was actually used to fill the hole. Crucially, we export the extracted problems not only as strings, but also as structures; the export preserves and specifies all type information available to the checker, including references and token structure at the subtype level. We design our neural model so as to effectively maintain and utilize this information; this is what the types of ℕ, + and +-comm look like post-tokenization:



Results. Extensive evaluation shows the model to perform adequately in a realistic premise selection setup, where it in fact outperforms several traditional (structure-oblivious) alternatives of the same scale.

For details, we redirect the interested reader to the full published manuscript [1].

References

1. Kogkalidis, K., Melkonian O., Bernardy, J.-P.: Learning Structure-Aware Representations of Dependent Types.
2. AGDA2TRAIN Github repository: https://github.com/omelkonian/agda2train.
3. QUILL Github repository: https://github.com/konstantinosKokos/quill/.