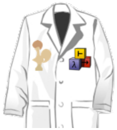
Learning to Format Coq Code Using Language Models

Pengyu Nie[1], **Karl Palmskog**[2],
Junyi Jessy Li[1], and Milos Gligoric[1]

The Coq Workshop 2020



[1] The University of Texas at Austin

[2] KTH Royal Institute of Technology

Background: Coq is a Language Platform

Coq extensibility has provided us with a linguistic zoo:

- libraries: MathComp, Stdpp, TLC, Stdlib, ...
- tactic and proof languages: Ltac, Ltac2, Mtac2, SSReflect, ...
- embedded languages: Verifiable C, RustBelt, MetaCoq, ...

Example: Coq/SSReflect/MathComp

```
Lemma totient_coprime m n :
  coprime m n -> totient (m * n) = totient m * totient n.
Proof.
move=> co_mn; have [-> //| m_gt0] := posnP m.
have [->|n_gt0] := posnP n; first by rewrite !muln0.
rewrite !totientE ?muln_gt0 ?m_gt0 //.
have /(perm_big _)->: perm_eq (primes (m * n)) (primes m ++ primes n).
  apply: uniq_perm => [||p]; first exact: primes_uniq.
    by rewrite cat_uniq !primes_uniq -coprime_has_primes // co_mn.
  by rewrite mem_cat primes_mul.
rewrite big_cat /= !big_seq.
congr (_ * _); apply: eq_bigr => p; rewrite mem_primes => /and3P[_ _ dvp].
  rewrite (mulnC m) logn_Gauss //; move: co_mn.
  by rewrite -(divnK dvp) coprime_mull => /andP[].
rewrite logn_Gauss //; move: co_mn.
by rewrite coprime_sym -(divnK dvp) coprime_mull => /andP[].
Qed.
```

Example: Coq/Ltac/Stdpp

```
Lemma list_find_app_Some l1 l2 i x :
  list_find P (l1 ++ l2) = Some (i,x) ↔
    list_find P l1 = Some (i,x) ∨
    length l1 ≤ i ∧ list_find P l1 = None ∧ list_find P l2 = Some (i - length l1,x).
Proof.
  split.
  - intros ([?|[??]]%lookup_app_Some&?&Hleast)%list_find_Some.
    + left. apply list_find_Some; eauto using lookup_app_l_Some.
    + right. split; [lia|]. split.
      { apply list_find_None, Forall_lookup. intros j z ??.
        assert (j < length l1) by eauto using lookup_lt_Some.
        naive_solver eauto using lookup_app_l_Some with lia. }
      apply list_find_Some. split_and!; [done..|].
      intros j z ??. eapply (Hleast (length l1 + j)); [|lia].
      by rewrite lookup_app_r, minus_plus by lia.
  - intros [(?&?&Hleast)%list_find_Some|(?&Hl1&(?&?&Hleast)%list_find_Some)].
    + apply list_find_Some. split_and!; [by auto using lookup_app_l_Some..|].
      assert (i < length l1) by eauto using lookup_lt_Some.
      intros j y ?%lookup_app_Some; naive_solver eauto with lia.
    + rewrite list_find_Some, lookup_app_Some. split_and!; [by auto..|].
      intros j y [?|?]%lookup_app_Some ?; [|naive_solver auto with lia].
      by eapply (Forall_lookup_1 (not ∘ P) l1); [by apply list_find_None|..].
Qed.
```

Example: Coq/Ltac/Stdlib

```
Lemma sec_left_sum_tree (X Y:Set) (p : WFT X):
  forall (A : X -> X -> Prop), SecureBy A p ->
  SecureBy (left_sum_lift A) (left_sum_tree Y p).
induction p.
  intros A Zsec.
  simpl in *. intros v w x y z.
  destruct x; (repeat (auto; firstorder)).
  destruct v; (repeat (auto; firstorder)).
  destruct w; (repeat (auto; firstorder)).
  destruct v; (repeat (auto; firstorder)).
  destruct w; (repeat (auto; firstorder)).
  intros. simpl. intro x. destruct x; repeat auto.
  eapply sec_strengthen. Focus 2. apply H. apply H0.
  intros. destruct x0; repeat (auto; firstorder).
          destruct y; repeat (auto; firstorder).
  simpl in *. intro x.
  destruct x; repeat (auto; firstorder).
  eapply sec_strengthen. Focus 2. apply H. apply H0.
  intros. destruct x0; repeat (auto;firstorder).
          destruct y0; repeat (auto;firstorder).
Defined.
```

Problem: Users Need Help to Follow Coding Conventions

- coding conventions are important in large/medium sized Coq projects
- but, writing fully idiomatic Coq/SSReflect takes months of training ...
- ... and doesn't generalize to projects using Stdpp or CompCert
- reading contribution guidelines is no substitute for expert feedback!

Enforcing Conventions: Coq's Beautifier (make beautify)

```
Lemma sec_left_sum_tree (X Y : Set) (p : WFT X) :
  forall A : X -> X -> Prop,
  SecureBy A p ->
  SecureBy (left_sum_lift
             A) (left_sum_tree Y p).(induction
  p).(
   intros A Zsec).(
   simpl in *).( intros v w x y z).(destruct x; repeat (auto;
                      firstorder)).(destruct v; repeat (auto;
                      firstorder)).(destruct w; repeat (auto;
                      firstorder)).(destruct v; repeat (auto;
                      firstorder)).(destruct w; repeat (auto;
                      firstorder)).(
   intros).( simpl). intro x.(destruct x; repeat auto).(eapply
    sec_strengthen).Focus 2.(apply H).(apply H0).(
   intros).(destruct x0; repeat (auto; firstorder)).(destruct y; repeat (auto;
                              firstorder)).(
   simpl in *). intro x.(destruct x; repeat (auto;
                      firstorder)).(eapply
    sec_strengthen).Focus 2.(apply H).(apply H0).(
   intros).(destruct x0; repeat (auto; firstorder)).(destruct y0; repeat (auto;
                               firstorder)).Defined.
```

Enforcing Conventions: SerAPI's Pretty-Printer

```
Lemma sec_left_sum_tree (X Y : Set) (p : WFT X) :
  forall A : X -> X -> Prop,
  SecureBy A p -> SecureBy (left_sum_lift A) (left_sum_tree Y p).
(induction p).
(intros A Zsec).
(simpl in *).
(intros v w x y z).
(destruct x; repeat (auto; firstorder)).
(destruct v; repeat (auto; firstorder)).
(destruct w; repeat (auto; firstorder)).
(destruct v; repeat (auto; firstorder)).
(destruct w; repeat (auto; firstorder)).
(intros).
(simpl).
intro x.
(destruct x; repeat auto).
(eapply sec_strengthen).
Focus 2.
(apply H).
(apply H0).
(intros).
(destruct x0; repeat (auto; firstorder)).
(destruct y; repeat (auto; firstorder)).
(* ... more of the same ... *)
Defined.
```

Pros and Cons of Rule-Based Linting

+ simple and fast
+ easy to integrate into development process
- addresses small subset of all conventions
- tedious to define new rules
- will never support all Coq languages

A Flexible Alternative: Naturalness and Language Models

- Coq code has high **naturalness**, i.e., repetitions and patterns
- naturalness of code can be exploited in **language models**
- language models summarize statistical properties of code
- there are already Java formatters/analyzers using naturalness

Our Message to the Coq Community

- rule-based linters will always lag behind prevailing conventions
- language models are the **right way** to handle conventions:
  1. pick a trained language model based on preferred library/style
  2. refine the model by training it on your own code
  3. use refined model to suggest conventions in all code
- rule-based linters still useful as **rerankers** of suggestions

Our Contributions

- two initial language models to learn and suggest **space formatting** in Coq files: baseline and advanced
- implementation of the language models in a toolchain based on Coq 8.10 and SerAPI 0.7.1
- preliminary evaluation using a MathComp 1.9.0 based corpus
  - machine readable representations as S-expressions via SerAPI
  - 100k+ proof script lines, 63k+ lines of Gallina
  - 2.2M+ Coq lexer tokens
- this is part of an **umbrella project** to suggest coding conventions for Coq using machine learning techniques
  - `https://github.com/EngineeringSoftware/roosterize`

Running Example From the RegLang Project

```
Lemma mg_eq_proof L1 L2 (N1 : mgClassifier L1) : L1 =i L2 -> nerode L2 N1.
Proof. move => H0 u v. split => [/nerodeP H1 w|H1].
  - by rewrite -!H0.
  - apply/nerodeP => w. by rewrite !H0.
Qed.
```

Machine Learning Approach

Task: predict spacing between tokens obtained from Coq's lexer

1. obtain tokens and spacing via SerAPI's `sertok` program
2. train model on spacing between tokens in lots of Coq code
3. use model to predict spacing between two given Coq tokens

Feature Extraction

```
Lemma mg_eq_proof L1 L2 (N1 : mgClassifier L1) : L1 =i L2 -> nerode L2 N1.
```

```
(Sentence((IDENT Lemma)(IDENT mg_eq_proof)(IDENT L1)(IDENT L2)
  (KEYWORD"(")(IDENT N1)(KEYWORD :)(IDENT mgClassifier)
  (IDENT L1)(KEYWORD")")(KEYWORD :)(IDENT L1)(KEYWORD =i)(IDENT L2)
  (KEYWORD ->)(IDENT nerode)(IDENT L2)(IDENT N1)(KEYWORD .)))
```

$$(\texttt{Content}, \texttt{Kind}, \#\texttt{Newlines}, \#\texttt{Spaces})$$

$$[(\texttt{null}, \texttt{BOS}, 0, 0), (\texttt{Lemma}, \texttt{IDENT}, 2, 0), (\texttt{mg\_eq\_proof}, \texttt{IDENT}, 0, 1), \ldots]$$

Language Models: n-gram and Neural

n-gram Model (Baseline)

- inserts spacing as special tokens before each token
- predicts next token after observing the $n-1$ previous ones by **statistical** way (finding the most frequent token appearing after the $n-1$ previous tokens in the training set)

Neural Model (Advanced)

- embeds Coq tokens and spacing information into vectors
- predicts spacing using embedding vectors
- captures **deeper** formatting rules than statistical approach

Corpus Based on MathComp 1.9.0

Project	SHA	#Files	#Lemmas	#Toks	LOC	
					Spec.	Proof
finmap	27642a8	4	940	78,449	4,260	2,191
fourcolor	0851d49	60	1,157	560,682	9,175	27,963
math-comp	748d716	89	8,802	1,076,096	38,243	46,470
odd-order	ca602a4	34	367	519,855	11,882	24,243
Avg.	N/A	46.75	2,816.50	558,770.50	15,890.00	25,216.75
Σ	N/A	187	11,266	**2,235,082**	**63,560**	**100,867**

Evaluation Setup

1. Randomly split corpus files into training, validation and testing sets which contain 80%, 10%, 10% of the files, respectively
2. Train model using training and validation sets
3. Apply model on testing set, and evaluate suggested spacing against existing spacing

Results

Model	Top-1 Accuracy	Top-3 Accuracy
Neural	**96.8%**	**99.7%**
n-gram	93.4%	98.9%

Caveats:

- top-k accuracy assumes all errors are equally important
- but, subjective severity of spacing errors can differ greatly

Highlight in Neural Formatting of `polydiv.v`

```
(* neural model formatting *)
Definition redivp_rec (q : {poly R}) :=
  let sq := size q in
  let cq := lead_coef q in
  fix loop (k : nat) (qq r : {poly R}) (n : nat) {struct n} :=
  if size r < sq then (k, qq, r) else
  let m := (lead_coef r) *: 'X^(size r - sq) in
  let qq1 := qq * cq%:P + m in
  let r1 := r * cq%:P - m * q in
  if n is n1.+1 then loop k.+1 qq1 r1 n1 else (k.+1, qq1, r1).

(* manual formatting *)
Definition redivp_rec (q : {poly R}) :=
  let sq := size q in
  let cq := lead_coef q in
   fix loop (k : nat) (qq r : {poly R})(n : nat) {struct n} :=
    if size r < sq then (k, qq, r) else
    let m := (lead_coef r) *: 'X^(size r - sq) in
    let qq1 := qq * cq%:P + m in
    let r1 := r * cq%:P - m * q in
       if n is n1.+1 then loop k.+1 qq1 r1 n1 else (k.+1, qq1, r1).
```

Problem: Line Breaks Before Qed

```
Lemma rdivp_small p q : size p < size q -> rdivp p q = 0.
Proof.
rewrite /rdivp unlock; have [-> | _ ltpq] := eqP; first by rewrite size_poly0.
by case: (size p) => [|s]; rewrite /= ltpq. Qed.

(* ... *)

Lemma eq_rdvdp q1 p : p = q1 * d -> rdvdp d p.
Proof.
move=> h; apply: (@eq_rdvdp _ _ _ _ 1%N q1); rewrite (eqP mond).
- exact: commr1.
- exact: rreg1. by rewrite expr1n mulr1. Qed.
```

Problem: Indenting Proof Scripts

```
(* neural model formatting *)
Lemma ltn_rmodp p q : (size (rmodp p q) < size q) = (q != 0).
Proof.
rewrite /rdivp /rmodp /rscalp unlock; case q0: (q == 0).
  by rewrite (eqP q0) /= size_poly0 ltn0.
elim: (size p) 0%N 0 {1 3}p (leqnn (size p)) => [|n ihn] k q1 r.
rewrite leqn0 size_poly_eq0; move/eqP->; rewrite /= size_poly0 /= lt0n.
by rewrite size_poly_eq0 q0 /= size_poly0 lt0n size_poly_eq0 q0.
move=> hr /=; case: (@ltnP (size r) _) => //= hsrq; rewrite ihn //.
apply/leq_sizeP=> j hnj; rewrite coefB.

(* manual formatting *)
Lemma ltn_rmodp p q : (size (rmodp p q) < size q) = (q != 0).
Proof.
rewrite /rdivp /rmodp /rscalp unlock; case q0 : (q == 0).
  by rewrite (eqP q0) /= size_poly0 ltn0.
elim: (size p) 0%N 0 {1 3}p (leqnn (size p)) => [|n ihn] k q1 r.
  rewrite leqn0 size_poly_eq0; move/eqP->; rewrite /= size_poly0 /= lt0n.
  by rewrite size_poly_eq0 q0 /= size_poly0 lt0n size_poly_eq0 q0.
move=> hr /=; case: (@ltnP (size r) _) => //= hsrq; rewrite ihn //.
apply/leq_sizeP => j hnj; rewrite coefB.
```

Next Steps

- What metrics should we use to judge spacing correctness?
- What are the main sets of coding conventions for Coq?
- How can we define rule-based formatting abstractly?
- Can formatting be combined with refactoring?
- How should formatting suggestions be integrated into Coq-based development processes?

Conclusions

- Language models are the solution to user support for coding conventions
- Simple models for spacing already provide 96%+ top-1 accuracy
- Rule-based linters can complement language models (reranking)
- But, language models need proper training and evaluation
- Help us (on Coq's Zulip chat) to curate datasets and metrics!

MathComp corpus: `https://github.com/EngineeringSoftware/math-comp-corpus`

Pengyu Nie (pynie@utexas.edu) Karl Palmskog (palmskog@kth.se)

TEXAS
The University of Texas at Austin

KTH
VETENSKAP OCH KONST