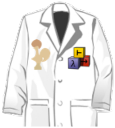
# Deep Generation of Coq Lemma Names Using Elaborated Terms

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

IJCAR 2020



[1] The University of Texas at Austin

[2] KTH Royal Institute of Technology

# Motivation: Verification Projects Growing in Size

- Proof assistants are increasingly used to formalize results in advanced mathematics and develop large trustworthy software systems

| **Project** | **Domain** | **Assistant** | **LOC** |
|---|---|---|---:|
| CompCert | compiler | Coq | 120k+ |
| MathComp | math | Coq | 85k+ |
| Verdi Raft | k/v store | Coq | 50k+ |
| seL4 | kernel | Isabelle/HOL | 200k+ |
| BilbyFS | file system | Isabelle/HOL | 14k+ |

- Verification projects face challenges similar to those in large software projects: maintenance and enforcement of coding conventions

# Motivation: Verification Projects Growing in Size

- Proof assistants are increasingly used to formalize results in advanced mathematics and develop large trustworthy software systems

| Project | Domain | Assistant | LOC |
|---|---|---|---|
| CompCert | compiler | Coq | 120k+ |
| MathComp | math | Coq | 85k+ |
| Verdi Raft | k/v store | Coq | 50k+ |
| seL4 | kernel | Isabelle/HOL | 200k+ |
| BilbyFS | file system | Isabelle/HOL | 14k+ |

- Verification projects face challenges similar to those in large software projects: maintenance and enforcement of coding conventions
- **How to name lemmas?**

# Motivation: Hard-coded Naming Conventions

## CONTRIBUTIONS.md in MathComp, 50+ entries

### Naming conventions for lemmas (non exhaustive)

**Names in the library usually obey one of the following conventions**

- `(condition_)?mainSymbol_suffixes`
- `mainSymbol_suffixes(_condition)?` Or in the presence of a property denoted by an n-ary or unary predicate:
- `naryPredicate_mainSymbol+`
- `mainSymbol_unaryPredicate`

**Where:**

- `mainSymbol` is the most meaningful part of the lemma. It generally is the head symbol of the right-hand side of an equation or the head symbol of a theorem. It can also simply be the main object of study, head symbol or not. It is usually either
  - one of the main symbols of the theory at hand. For example, it will be `opp`, `add`, `mul`, etc., or
  - a special "canonical" operation, such as a ring morphism or a subtype predicate. e.g. `linear`, `raddf`, `rmorph`, `rpred`, etc.
- "condition" is used when the lemma applies under some hypothesis.
- "suffixes" are there to refine what shape and/or what other symbols the lemma has. It can either be the name of a symbol ("add", "mul", etc.), or the (short) name of a predicate ("`inj`" for "`injectivity`", "`id`" for "identity", etc.) or an abbreviation. Abbreviations are in the header of the file which introduces them. We list here the main abbreviations.
- `A` -- associativity, as in `andbA : associative andb.`
- `AC` -- right commutativity.
- `ACA` -- self-interchange (inner commutativity), e.g., `orbACA : (a || b) || (c || d) = (a || c) || (b || d).`
- `b` -- a boolean argument, as in `andbb : idempotent andb.`
- `C` -- commutativity, as in `andbC : commutative andb.` -- alternatively, predicate or set complement, as in `predC.`

# Motivation: Many Inconsistencies in Large Projects

math-comp / **math-comp**

Code | Issues 69 | Pull requests 29 | Actions | Projects | Wiki | Security | Insights

## mem_imset mem_map naming/statement inconsistency #508

**Open** **chdoc** opened this issue on 14 May · 9 comments

**chdoc** commented on 14 May — Contributor

while cleaning, I noticed the following oddity:

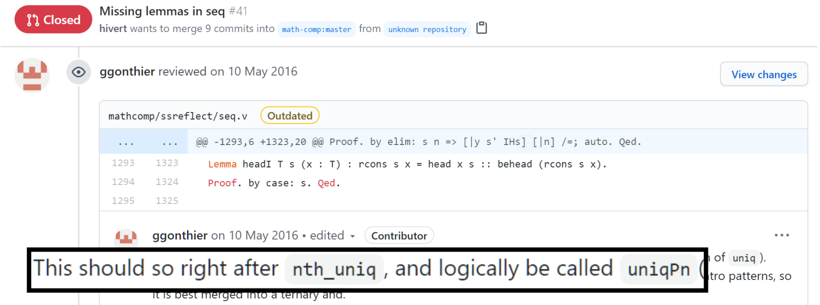
```
Lemma mem_imset (aT rT : finType) (f : aT -> rT) (D : {pred aT}) (x : aT) :
  x \in D -> f x \in [set f x | x in D]

Lemma map_f (T1 T2 : eqType) (f : T1 -> T2) (s : seq T1) (x : T1) :
   x \in s -> f x \in [seq f i | i <- s]
Lemma mem_map (T1 T2 : eqType) (f : T1 -> T2),
  injective f -> forall (s : seq T1) (x : T1), (f x \in [seq f i | i <- s]) = (x \in s)
```

Wouldn't it be more consistent to rename `mem_imset` to `imset_f`

```
Lemma mem_imset (aT rT : finType) (f : aT -> rT) (A : {set aT}) (x : aT) :
  injective f -> (f x \in f @: A) = (x \in A).
```

1

# Motivation: Manually Checking and Enforcing



# Our Contributions

- **Roosterize**: toolchain for learning and suggesting lemma names
  - Code review process
  - Interactive development
  - Batch mode

# Our Contributions

- **Roosterize**: toolchain for learning and suggesting lemma names
  - Code review process
  - Interactive development
  - Batch mode

- Novel **generation models** based on multi-input encoder-decoder neural networks leveraging **elaborated terms**

# Our Contributions

- **Roosterize**: toolchain for learning and suggesting lemma names
  - Code review process
  - Interactive development
  - Batch mode
- Novel **generation models** based on multi-input encoder-decoder neural networks leveraging **elaborated terms**
- A **corpus** of 164k LOC high quality Coq code

# Our Contributions

- **Roosterize**: toolchain for learning and suggesting lemma names
  - Code review process
  - Interactive development
  - Batch mode
- Novel **generation models** based on multi-input encoder-decoder neural networks leveraging **elaborated terms**
- A **corpus** of 164k LOC high quality Coq code
- An extensive **evaluation** on our corpus via automated metrics

# Our Contributions

- **Roosterize**: toolchain for learning and suggesting lemma names
  - Code review process
  - Interactive development
  - Batch mode
- Novel **generation models** based on multi-input encoder-decoder neural networks leveraging **elaborated terms**
- A **corpus** of 164k LOC high quality Coq code
- An extensive **evaluation** on our corpus via automated metrics
- A qualitative **case study** on a project outside corpus

# Running Example: A Lemma from reglang Project

- A lemma from a project on the theory of regular languages
- **M**ost **g**eneral classifiers can be casted to **eq**uivalent languages

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

# Running Example: A Lemma from reglang Project

- A lemma from a project on the theory of regular languages
- **M**ost **g**eneral classifiers can be casted to **eq**uivalent languages

Lemma name

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

# Running Example: A Lemma from `reglang` Project

- A lemma from a project on the theory of regular languages
- **M**ost **g**eneral classifiers can be casted to **eq**uivalent languages

Lemma statement

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

# Running Example: A Lemma from reglang Project

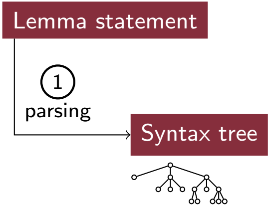
- A lemma from a project on the theory of regular languages
- **M**ost **g**eneral classifiers can be casted to **eq**uivalent languages

Proof script

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

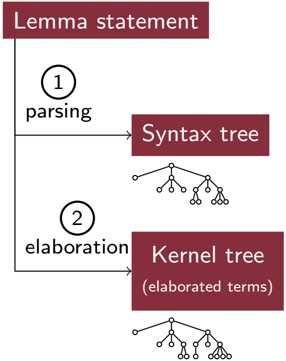
# Roosterize Toolchain

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



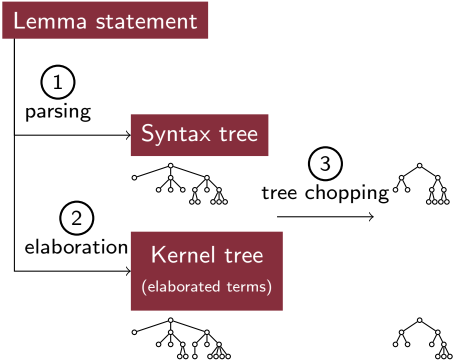
# Roosterize Toolchain

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



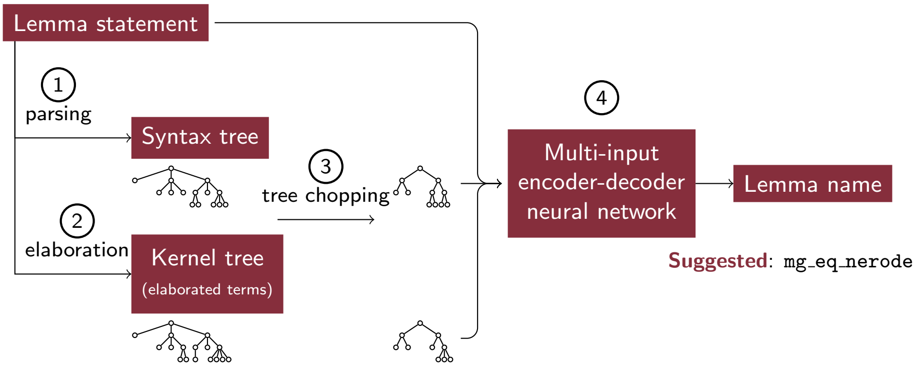
# Roosterize Toolchain

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



# Roosterize Toolchain

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



# Model Input: Lemma Statement

```
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 .)))
```

S-expression

- In lexing phase
- **Surface syntax** level information

# Model Input: Syntax Tree

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

```
(VernacExpr()(VernacStartTheoremProof Lemma (Id mg_eq_proof)
 (((CLocalAssum(Name(Id L1))(CLocalAssum(Name(Id L2)))
   (CLocalAssum(Name(Id N1))(CApp(CRef(Ser_Qualid(DirPath())(Id mgClassifier)))
    (CRef(Ser_Qualid(DirPath())(Id L1))))))
  (CNotation(InConstrEntrySomeLevel"_ -> _")
   (CNotation(InConstrEntrySomeLevel"_ =i _")
    (CRef(Ser_Qualid(DirPath())(Id L1)))(CRef(Ser_Qualid(DirPath())(Id L2))))
   (CApp(CRef(Ser_Qualid(DirPath())(Id nerode)))
    (CRef(Ser_Qualid(DirPath())(Id L2)))(CRef(Ser_Qualid(DirPath())(Id N1))))))))
```

- In parsing phase
- **Surface syntax** level information

# Model Input: Kernel Tree

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

```
(Prod (Name (Id char)) ... (Prod (Name (Id L1)) ...
  (Prod (Name (Id L2)) ... (Prod (Name (Id N1)) ...
   (Prod Anonymous (App (Ref (DirPath ((Id ssrbool) (Id ssr) (Id Coq))) (Id eq_mem)) ...
     (Var (Id L1)) ... (Var (Id L2)))
    (App (Ref (DirPath ((Id myhill_nerode) (Id RegLang))) (Id nerode)) ...
     (Var (Id L2)) ... (Var (Id N1))))))))
```

- In elaboration phase
- **Semantic** level information

# Model Input: Kernel Tree

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

```
(Prod (Name (Id char)) ... (Prod (Name (Id L1)) ...
  (Prod (Name (Id L2)) ... (Prod (Name (Id N1)) ...
   (Prod Anonymous (App (Ref (DirPath ((Id ssrbool) (Id ssr) (Id Coq))) (Id eq_mem)) ...
     (Var (Id L1)) ... (Var (Id L2)))
    (App (Ref (DirPath ((Id myhill_nerode) (Id RegLang))) (Id nerode)) ...
     (Var (Id L2)) ... (Var (Id N1))))))))
```

- In elaboration phase
- **Semantic** level information
  - Add implicit terms

# Model Input: Kernel Tree

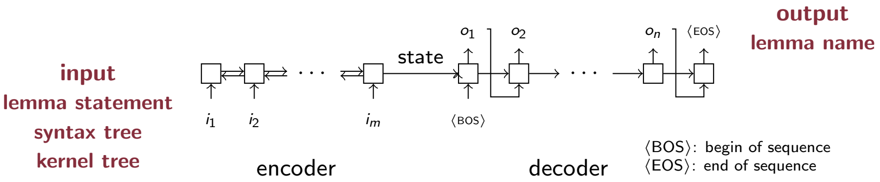
```
Lemma mg_eq_proof L1 L2 (N1 : mgClassifier L1) : L1 =i L2 -> nerode L2 N1.
```

```
(Prod (Name (Id char)) ... (Prod (Name (Id L1)) ...
  (Prod (Name (Id L2)) ... (Prod (Name (Id N1)) ...
   (Prod Anonymous (App (Ref (DirPath ((Id ssrbool) (Id ssr) (Id Coq))) (Id eq_mem)) ...
     (Var (Id L1)) ... (Var (Id L2)))
    (App (Ref (DirPath ((Id myhill_nerode) (Id RegLang))) (Id nerode)) ...
     (Var (Id L2)) ... (Var (Id N1))))))))
```

- In elaboration phase
- **Semantic** level information
  - Add implicit terms
  - Translate operators to their kernel names

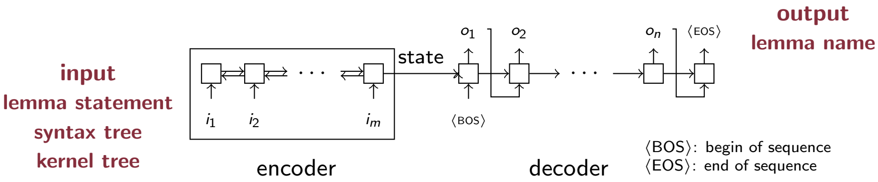
# Lemma Naming as a Transduction Task

- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)



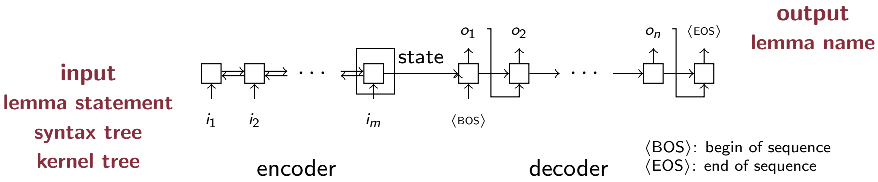
# Lemma Naming as a Transduction Task

- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)



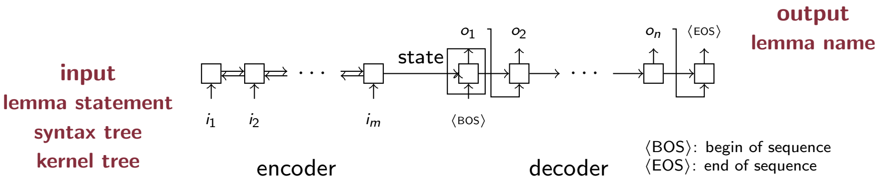
# Lemma Naming as a Transduction Task

- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)



# Lemma Naming as a Transduction Task

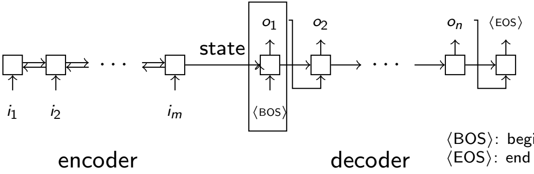
- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)



# Lemma Naming as a Transduction Task

- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)

**input**
**lemma statement**
**syntax tree**
**kernel tree**

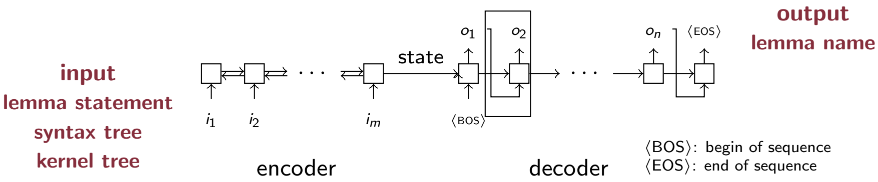



**output**
**lemma name**

⟨BOS⟩: begin of sequence
⟨EOS⟩: end of sequence

# Lemma Naming as a Transduction Task

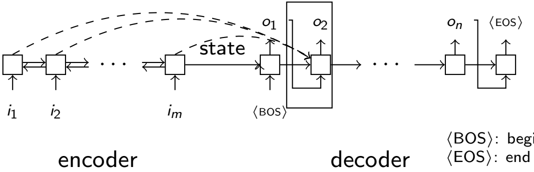
- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)



# Lemma Naming as a Transduction Task

- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)
- **Attention mechanism**: decoder can "pay attention to" different parts of the inputs at each time step

**input**
**lemma statement**
**syntax tree**
**kernel tree**



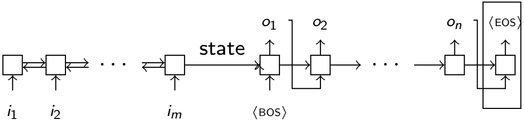

**output**
**lemma name**

⟨BOS⟩: begin of sequence
⟨EOS⟩: end of sequence

# Lemma Naming as a Transduction Task

- **Encoder-decoder neural network**: specifically designed for **transduction tasks** (e.g., machine translation, summarization, question answering)
- **Attention mechanism**: decoder can "pay attention to" different parts of the inputs at each time step

**input**
**lemma statement**
**syntax tree**
**kernel tree**



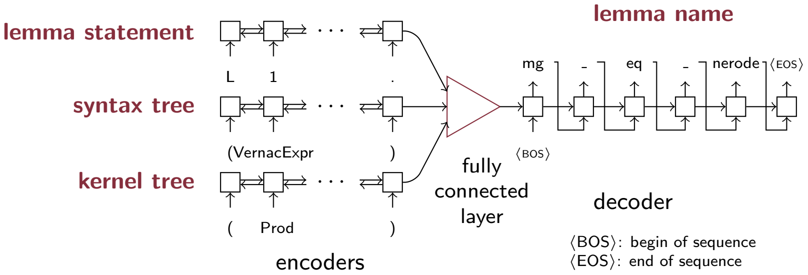

**output**
**lemma name**

encoder

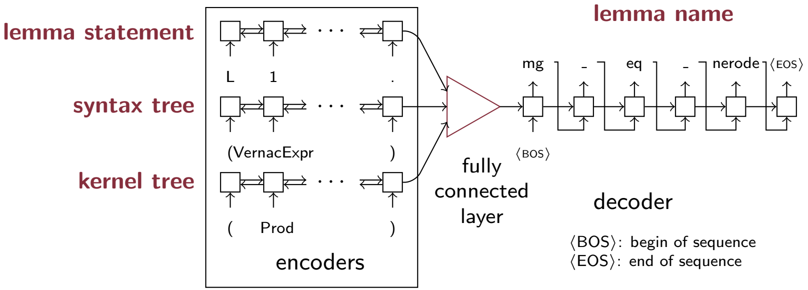
decoder

⟨BOS⟩: begin of sequence
⟨EOS⟩: end of sequence

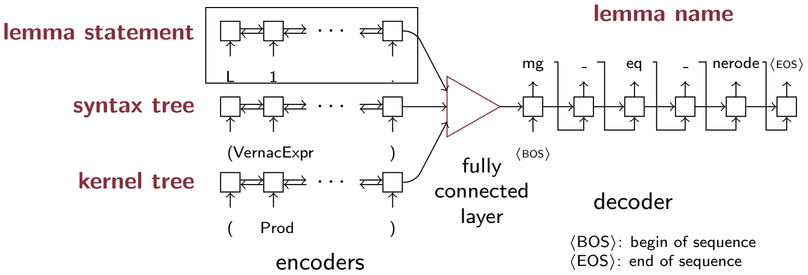
# Multi-input Encoder-decoder Neural Network Architecture



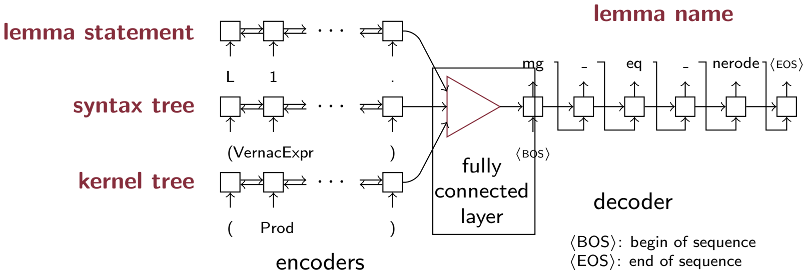
# Multi-input Encoder-decoder Neural Network Architecture



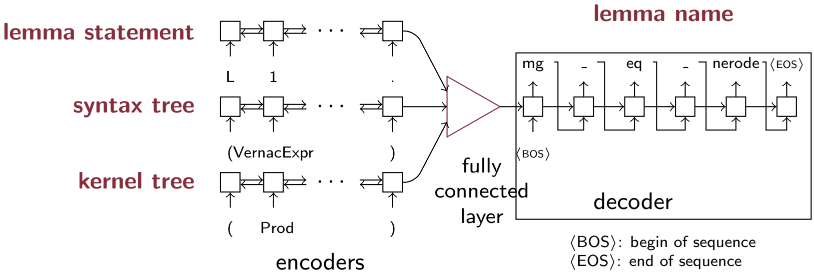
# Multi-input Encoder-decoder Neural Network Architecture



# Multi-input Encoder-decoder Neural Network Architecture

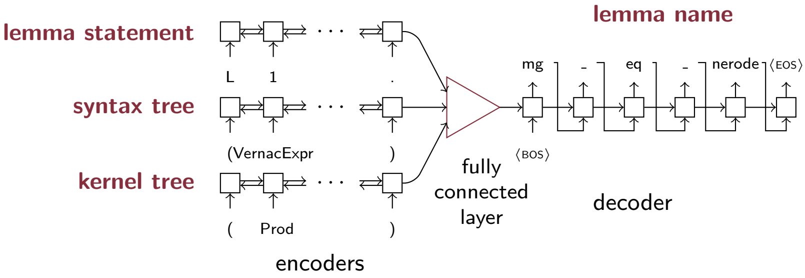


# Multi-input Encoder-decoder Neural Network Architecture



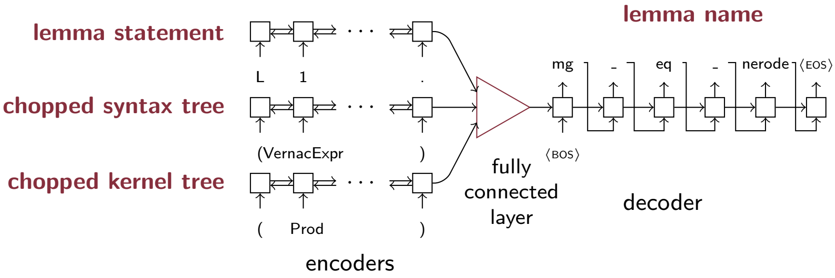

⟨BOS⟩: begin of sequence
⟨EOS⟩: end of sequence

# Tree Chopping




- Syntax and kernel trees can be large, which prevents the neural networks to learn effectively
- Some parts are irrelevant for naming and can be “chopped”

# Tree Chopping




- Syntax and kernel trees can be large, which prevents the neural networks to learn effectively
- Some parts are irrelevant for naming and can be "chopped"
- Tree chopping heuristics:
  1. Replace the fully qualified name sub-trees with only the last component of the name
  2. Remove the location information
  3. Extract the singletons

# Example Tree Chopping

- Before chopping

```
(Prod Anonymous (App (Ref (DirPath ((Id ssrbool) (Id ssr) (Id Coq))) (Id eq_mem))
  ... ((App (Ref ... ))) ... ))
```

# Example Tree Chopping

- Before chopping

**#1 prefixes in a fully-qualified name**:
usually related to directory paths and likely not relevant

```
(Prod Anonymous (App (Ref (DirPath ((Id ssrbool) (Id ssr) (Id Coq))) (Id eq_mem))
  ... ((App (Ref ... ))) ... ))
```

**#3 singleton**: unnecessarily increase tree size

# Example Tree Chopping

- Before chopping

**#1 prefixes in a fully-qualified name**:
usually related to directory paths and likely not relevant

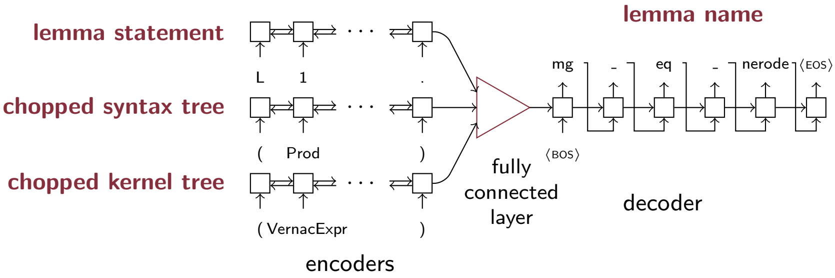
```
(Prod Anonymous (App (Ref (DirPath ((Id ssrbool) (Id ssr) (Id Coq))) (Id eq_mem))
  ... ((App (Ref ... ))) ... ))
```

**#3 singleton**: unnecessarily increase tree size

- After chopping

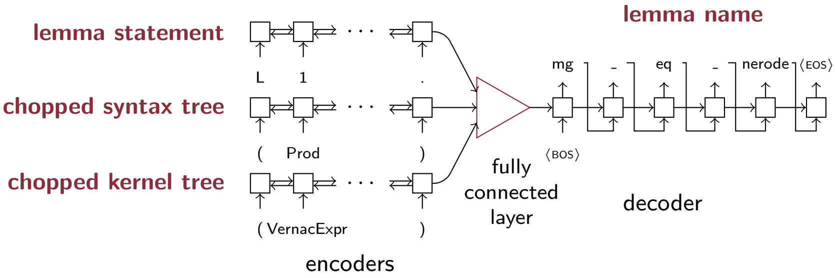
```
(Prod Anonymous (App eq_mem ... (App (Ref ... )) ... ))
```

# Sub-tokenization



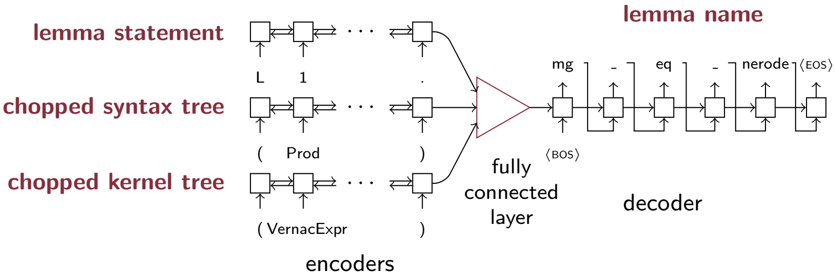

- Coq names have multiple components (e.g., prefixes and suffixes), making the vocabulary large and sparse

# Sub-tokenization




- Coq names have multiple components (e.g., prefixes and suffixes), making the vocabulary large and sparse
- All inputs and outputs are **sub-tokenized** (e.g., `extprod_mulgA` → `extprod`, `_`, `mul`, `g`, and `A`)

# Sub-tokenization




- Coq names have multiple components (e.g., prefixes and suffixes), making the vocabulary large and sparse
- All inputs and outputs are **sub-tokenized** (e.g., `extprod_mulgA` → `extprod`, `_`, `mul`, `g`, and `A`)
- Reduces the sparsity of the vocabulary and improves the performance of the model

# Corpus: MathComp Family of Projects

- We constructed a corpus of four large Coq projects from the MathComp family, totaling **164k lines of code**
- High quality and stringent adherence to coding conventions

| Project | SHA | #Files | #Lemmas | #Toks | LOC Spec. | LOC 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

- Randomly split corpus files into training, validation and testing sets which contain 80%, 10%, 10% of the files, respectively

| | #Files | #Lemmas | Name | | Lemma Statement | |
|---|---|---|---|---|---|---|
| | | | #Char | #SubToks | #Char | #SubToks |
| training | 152 | **8,861** | 10.14 | 4.22 | 44.16 | 19.59 |
| validation | 18 | **1,085** | 9.20 | 4.20 | 38.28 | 17.30 |
| testing | 17 | **1,320** | 9.76 | 4.34 | 48.49 | 23.20 |

# Evaluation: Setup

- Randomly split corpus files into training, validation and testing sets which contain 80%, 10%, 10% of the files, respectively

| | #Files | #Lemmas | Name #Char | Name #SubToks | Lemma Statement #Char | Lemma Statement #SubToks |
|---|---|---|---|---|---|---|
| training | 152 | **8,861** | 10.14 | 4.22 | 44.16 | 19.59 |
| validation | 18 | **1,085** | 9.20 | 4.20 | 38.28 | 17.30 |
| testing | 17 | **1,320** | 9.76 | 4.34 | 48.49 | 23.20 |

- Train ROOSTERIZE using training and validation sets
- Apply ROOSTERIZE on testing set, and evaluate generated lemma names against the reference lemma names (as written by developers)

# Evaluation: Automated Metrics

- **BLEU**
- **Fragment accuracy**
- **Top-1 accuracy**
- **Top-5 accuracy**

# Evaluation: Automated Metrics

- **BLEU**: range 0–100, percentage of 1–4-grams overlap between the characters of the generated name and the reference name
- **Fragment accuracy**
- **Top-1 accuracy**
- **Top-5 accuracy**

$$\mathrm{BLEU}(\texttt{card\_Syl\_dvd}, \texttt{card\_Syl\_dvd}) = 100$$
$$\mathrm{BLEU}(\texttt{card\_Syl\_dvd}, \texttt{card\_dvd\_Syl}) = 81.9$$
$$\mathrm{BLEU}(\texttt{card\_Syl\_dvd}, \texttt{card\_dvd}) = 52.7$$
$$\mathrm{BLEU}(\texttt{card\_Syl\_dvd}, \texttt{Frattini\_arg}) = 14.7$$

# Evaluation: Automated Metrics

- **BLEU**: range 0–100, percentage of 1–4-grams overlap between the characters of the generated name and the reference name
- **Fragment accuracy**: accuracy of generated names on the fragment level (defined by splitting the name by "_")
- **Top-1 accuracy**
- **Top-5 accuracy**

# Evaluation: Automated Metrics

- **BLEU**: range 0–100, percentage of 1–4-grams overlap between the characters of the generated name and the reference name
- **Fragment accuracy**: accuracy of generated names on the fragment level (defined by splitting the name by "_")
- **Top-1 accuracy**: frequency of the reference name fully matches the generated name
- **Top-5 accuracy**

# Evaluation: Automated Metrics

- **BLEU**: range 0–100, percentage of 1–4-grams overlap between the characters of the generated name and the reference name
- **Fragment accuracy**: accuracy of generated names on the fragment level (defined by splitting the name by "_")
- **Top-1 accuracy**: frequency of the reference name fully matches the generated name
- **Top-5 accuracy**: frequency of the reference name is one of the top-5 generated names

# Evaluation: Results

- Key results: ROOSTERIZE significantly outperforms baselines
- Ablation studies:
  - **Tree chopping** effectively improves performance
  - ROOSTERIZE's tree chopping is better than variants
  - Using **kernel trees** in inputs effectively improves performance (i.e., **semantics** information helps naming)

# Evaluation: Key Results

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Roosterize | **47.2** | **24.9%** | **9.6%** | **18.0%** |
| Baseline neural network based model | 20.0 | 4.7% | 0.1% | 0.3% |
| Baseline retrieval-based model | 28.3 | 10.0% | 0.2% | 0.3% |

- Baseline neural network based model: using only lemma statement as input, w/o attention mechanism
- Baseline retrieval-based model: *details in the paper*

# Evaluation: Key Results

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Roosterize | **47.2** | **24.9%** | **9.6%** | **18.0%** |
| Baseline neural network based model | 20.0 | 4.7% | 0.1% | 0.3% |
| Baseline retrieval-based model | 28.3 | 10.0% | 0.2% | 0.3% |

- Baseline neural network based model: using only lemma statement as input, w/o attention mechanism
- Baseline retrieval-based model: *details in the paper*
- Roosterize, using **lemma statement and chopped kernel tree** as inputs, obtained the best performance
  - 20+ points in BLEU better than baselines
  - statistically significantly better than all other model variants

# Ablation Study: Tree Chopping

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| ChopKnlTree+attn+copy | 42.9 | 19.8% | 5.0% | 11.7% |
| KnlTree+attn+copy | 37.0 | 14.2% | 2.2% | 8.4% |
| ChopSynTree+attn+copy | 39.8 | 18.3% | 6.8% | 12.2% |
| SynTree+attn+copy | 31.0 | 10.8% | 2.8% | 6.1% |

- **Tree chopping** improves performance by 6 points in BLEU for kernel tree and 9 points in BLEU for syntax tree

# Ablation Study: Tree Chopping

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| ChopKnlTree+attn+copy | 42.9 | 19.8% | 5.0% | 11.7% |
| KnlTree+attn+copy | 37.0 | 14.2% | 2.2% | 8.4% |
| ChopSynTree+attn+copy | 39.8 | 18.3% | 6.8% | 12.2% |
| SynTree+attn+copy | 31.0 | 10.8% | 2.8% | 6.1% |

- **Tree chopping** improves performance by 6 points in BLEU for kernel tree and 9 points in BLEU for syntax tree
- The size of the original trees and a lot of irrelevant data in those trees hurt the performance

# Ablation Study: Tree Chopping Variants

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Roosterize Chopping | 47.2 | 24.9% | 9.6% | 18.0% |
| Keep-category Chopping | 46.8 | 25.3% | 9.5% | 19.0% |
| Rule-based Chopping | 37.0 | 17.7% | 5.9% | 10.5% |
| Random Chopping | 37.7 | 19.2% | 6.7% | 10.9% |

# Ablation Study: Tree Chopping Variants

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Roosterize Chopping | 47.2 | 24.9% | 9.6% | 18.0% |
| Keep-category Chopping | 46.8 | 25.3% | 9.5% | 19.0% |
| Rule-based Chopping | 37.0 | 17.7% | 5.9% | 10.5% |
| Random Chopping | 37.7 | 19.2% | 6.7% | 10.9% |

- **Keep-category chopping** = Roosterize chopping, but keeps the category of referenced name in kernel trees, since that semantic information could be relevant for naming

# Ablation Study: Tree Chopping Variants

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Roosterize Chopping | 47.2 | 24.9% | 9.6% | 18.0% |
| Keep-category Chopping | 46.8 | 25.3% | 9.5% | 19.0% |
| Rule-based Chopping | 37.0 | 17.7% | 5.9% | 10.5% |
| Random Chopping | 37.7 | 19.2% | 6.7% | 10.9% |

- **Keep-category chopping** = Roosterize chopping, but keeps the category of referenced name in kernel trees, since that semantic information could be relevant for naming
- **Rule-based chopping** chops all nodes after depth 10, similar to the proof kernel tree processing heuristics used in ML4PG

# Ablation Study: Tree Chopping Variants

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Roosterize Chopping | 47.2 | 24.9% | 9.6% | 18.0% |
| Keep-category Chopping | 46.8 | 25.3% | 9.5% | 19.0% |
| Rule-based Chopping | 37.0 | 17.7% | 5.9% | 10.5% |
| Random Chopping | 37.7 | 19.2% | 6.7% | 10.9% |

- **Keep-category chopping** = Roosterize chopping, but keeps the category of referenced name in kernel trees, since that semantic information could be relevant for naming
- **Rule-based chopping** chops all nodes after depth 10, similar to the proof kernel tree processing heuristics used in ML4PG
- **Random chopping** chops random 91.4% nodes from the kernel tree to match the average number of nodes of Roosterize chopped trees, as the "dumb" baseline

# Ablation Study: Inputs

| Inputs Combinations | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Stmt+ChopKnlTree+ChopSynTree+attn+copy | 45.4 | 22.2% | 7.5% | 16.5% |
| Stmt+ChopKnlTree+attn+copy | **47.2** | **24.9%** | **9.6%** | **18.0%** |
| Stmt+ChopSynTree+attn+copy | 37.7 | 18.1% | 6.1% | 10.6% |
| ChopKnlTree+ChopSynTree+attn+copy | 45.4 | 22.9% | 7.6% | 15.3% |
| ChopKnlTree+attn+copy | 42.9 | 19.8% | 5.0% | 11.7% |
| ChopSynTree+attn+copy | 39.8 | 18.3% | 6.8% | 12.2% |
| Stmt+attn+copy | 38.9 | 19.4% | 6.9% | 11.6% |

- The inputs combination of **lemma statement and chopped kernel tree** works the best

# Ablation Study: Inputs

| Inputs Combinations | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Stmt+ChopKnlTree+ChopSynTree+attn+copy | 45.4 | 22.2% | 7.5% | 16.5% |
| Stmt+ChopKnlTree+attn+copy | **47.2** | **24.9%** | **9.6%** | **18.0%** |
| Stmt+ChopSynTree+attn+copy | 37.7 | 18.1% | 6.1% | 10.6% |
| ChopKnlTree+ChopSynTree+attn+copy | 45.4 | 22.9% | 7.6% | 15.3% |
| ChopKnlTree+attn+copy | 42.9 | 19.8% | 5.0% | 11.7% |
| ChopSynTree+attn+copy | 39.8 | 18.3% | 6.8% | 12.2% |
| Stmt+attn+copy | 38.9 | 19.4% | 6.9% | 11.6% |

- The inputs combination of **lemma statement and chopped kernel tree** works the best
- Lemma statement and syntax tree do not work well together because the two representations contain mostly the same information

# Ablation Study: Inputs

| Inputs Combinations | **BLEU** | **Frag.Acc.** | **Top-1** | **Top-5** |
|---|---|---|---|---|
| Stmt+ChopKnlTree+ChopSynTree+attn+copy | 45.4 | 22.2% | 7.5% | 16.5% |
| Stmt+ChopKnlTree+attn+copy | **47.2** | **24.9%** | **9.6%** | **18.0%** |
| Stmt+ChopSynTree+attn+copy | 37.7 | 18.1% | 6.1% | 10.6% |
| ChopKnlTree+ChopSynTree+attn+copy | 45.4 | 22.9% | 7.6% | 15.3% |
| ChopKnlTree+attn+copy | 42.9 | 19.8% | 5.0% | 11.7% |
| ChopSynTree+attn+copy | 39.8 | 18.3% | 6.8% | 12.2% |
| Stmt+attn+copy | 38.9 | 19.4% | 6.9% | 11.6% |

- The inputs combination of **lemma statement and chopped kernel tree** works the best
- Lemma statement and syntax tree do not work well together because the two representations contain mostly the same information
- Multiple inputs $\geq$ single input most of the times

# Case Study: Setup

- Motivation: generated lemma names may not match the manually written ones in the corpus, but can still be **semantically valid**, which is not reflected in our automated evaluation metrics

# Case Study: Setup

- Motivation: generated lemma names may not match the manually written ones in the corpus, but can still be **semantically valid**, which is not reflected in our automated evaluation metrics
- Apply ROOSTERIZE to a project outside of our corpus: the PCM library (#Files = 12, #Lemmas = 690)

# Case Study: Setup

- Motivation: generated lemma names may not match the manually written ones in the corpus, but can still be **semantically valid**, which is not reflected in our automated evaluation metrics
- Apply Roosterize to a project outside of our corpus: the PCM library (#Files = 12, #Lemmas = 690)
- Automated evaluation metrics: BLEU = 36.3, fragment accuracy = 17%, Top-1 accuracy = 5% (i.e., **36 lemmas** match exactly)

# Case Study: Setup

- Motivation: generated lemma names may not match the manually written ones in the corpus, but can still be **semantically valid**, which is not reflected in our automated evaluation metrics
- Apply ROOSTERIZE to a project outside of our corpus: the PCM library (#Files = 12, #Lemmas = 690)
- Automated evaluation metrics: BLEU = 36.3, fragment accuracy = 17%, Top-1 accuracy = 5% (i.e., **36 lemmas** match exactly)
- We asked the maintainer of the PCM library to evaluate **the remaining 654 lemma names** that do not match exactly and send us feedback

# Case Study: Findings

- The maintainer provided comments on 150 suggested names
- 20% were of good quality, out of which more than half were of high quality recall the analysis was of top-1 suggestions excluding exact matches

# Case Study: Findings

- The maintainer provided comments on 150 suggested names
- 20% were of good quality, out of which more than half were of high quality
  recall the analysis was of top-1 suggestions excluding exact matches
- Other suggested names tend to be “too generic”
- Unsuitable suggestions may contain useful parts

# Case Study: Examples

**Lemma statement**: `g e k v f : path ord k (supp f) ->`
`foldfmap g e (ins k v f) = g (k, v) (foldfmap g e f)`
**Hand-written**: `foldf_ins`
**Roosterize**: `foldfmap_ins`
**Comment**: ✓ The whole function name is used in the suggested name.

# Case Study: Examples

**Lemma statement**: `g e k v f : path ord k (supp f) ->`
`foldfmap g e (ins k v f) = g (k, v) (foldfmap g e f)`
**Hand-written**: `foldf_ins`
**Rooesterize**: `foldfmap_ins`
**Comment**: ✓ The whole function name is used in the suggested name.

**Lemma statement**: `: transitive (@ord T)`
**Hand-written**: `trans`
**Roosterize**: `ord_trans`
**Comment**: ✓ Useful to add the `ord` prefix to the name.

# Case Study: Examples

**Lemma statement**: `g e k v f : path ord k (supp f) ->`
`foldfmap g e (ins k v f) = g (k, v) (foldfmap g e f)`
**Hand-written**: `foldf_ins`
**Roosterize**: `foldfmap_ins`
**Comment**: ✓ The whole function name is used in the suggested name.

**Lemma statement**: `: transitive (@ord T)`
**Hand-written**: `trans`
**Roosterize**: `ord_trans`
**Comment**: ✓ Useful to add the `ord` prefix to the name.

**Lemma statement**: `p1 p2 s : kfilter (predI p1 p2) s =`
`kfilter p1 (kfilter p2 s)`
**Hand-written**: `kfilter_predI`
**Roosterize**: `eq_kfilter`
**Comment**: ✕ The suggested name is too generic.

# More Details in Our Paper

- Using copy mechanism to increase generalibility of models
- Using repetition prevention for decoder
- Implementation details of Roosterize toolchain
- Ablation study of more variants of Roosterize
- Expanded corpus of 21 MathComp family projects
- Generalizability case study: applying Roosterize on an out-of-corpus project with additional training

# Conclusions

- **Roosterize**: toolchain for learning and suggesting Coq lemma names, based on multi-input encoder-decoder neural networks
- **Kernel trees** provides important semantics context for lemma naming
- **Tree chopping** helps our models to effectively handle long inputs
- Evaluated on a **corpus** of 164k LOC high quality Coq code
- Case study shows ROOSTERIZE can provide useful suggestions in practice for a project outside our corpus

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

Pengyu Nie
pynie@utexas.edu

# Backup Slides After This Point

# Ablation Study: Copy Mechanism

| Model | BLEU | Frag.Acc. | Top-1 | Top-5 |
|---|---|---|---|---|
| Stmt+ChopKnlTree+attn+copy | **47.2** | **24.9%** | **9.6%** | **18.0%** |
| Stmt+ChopKnlTree+attn | 25.6 | 8.5% | 0.9% | 1.7% |

- **Copy mechanism** improves performance by 22 points in BLEU
- Many sub-tokens are specific to the file context and do not appear in the fixed vocabulary of the training set