

# Automation by Analogy, in Coq

Alasdair Hill, Katya Komendantskaya

Heriot-Watt University, Scotland

20 March 2018

# Machine Learning for Proof General (ML4PG)

ML4PG interfaces with proof general to extract features of lemmas from an ITP and uses a machine learning tool such as weka to cluster them.



## Feature Extraction

Feature extraction is performed to cluster lemmas on both proof terms and types

<sup>1</sup>Komendantskaya, E., Heras, J. and Grov, G., 2012. Machine learning in proof general: Interfacing interfaces. EPTCS 118 (User Interfaces for Theorem Provers), 15-41.

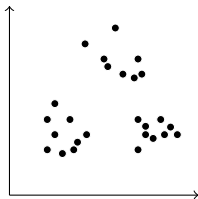
# ML4PG approach to proof-clustering

We have integrated Proof General with a variety of clustering algorithms:

# ML4PG approach to proof-clustering

We have integrated Proof General with a variety of clustering algorithms:

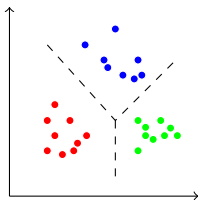
- Unsupervised machine learning technique:



# ML4PG approach to proof-clustering

We have integrated Proof General with a variety of clustering algorithms:

- Unsupervised machine learning technique:

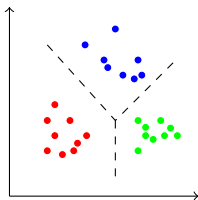


- Engines: Matlab, Weka, Octave, R, ...

# ML4PG approach to proof-clustering

We have integrated Proof General with a variety of clustering algorithms:

- Unsupervised machine learning technique:

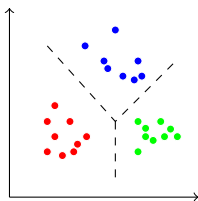


- Engines: [Matlab](#), [Weka](#), Octave, R, ...

# ML4PG approach to proof-clustering

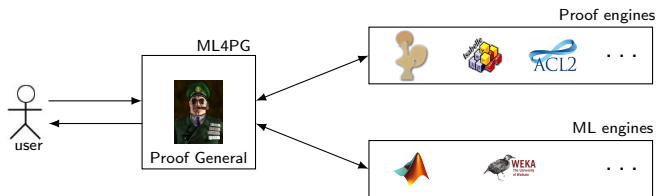
We have integrated Proof General with a variety of clustering algorithms:

- Unsupervised machine learning technique:



- Engines: [Matlab](#), [Weka](#), Octave, R, ...
- Algorithms: K-means, Gaussian Mixture models, simple Expectation Maximisation, ...

# Overall architecture of ML4PG



## Interaction with ML4PG:

- One interacts with Proof General as usual,
- when one cannot proceed with a proof,
- he calls ML4PG (command line or editor button),
- ML4PG informs the user of similar existing proofs/definitions.



# A proof in Coq with ML4PG help

The screenshot shows a Coq IDE window titled 'ml4pg.v'. The left pane contains the following Coq code:

```

end.

Lemma M1_corrected : forall l: list A, l = []
-> tl (tl (tl l) ++ nil) = nil.
Proof.
intro l.
intro H.
rewrite H.
rewrite app_nil_l2.
simpl; trivial.
Qed.

Lemma andb_false_r : forall (a : bool) , false = andb a false.
Proof.
intros.
case a.
  simpl; trivial.
simpl; trivial.
Qed.

Lemma M3_3b : forall (a: bool) (l :list bool), l = [a] -> andb (hdb [a]) fals
e = false.
Proof.
intros.

```

The right pane shows the current goal (ID 383) and its context:

```

1 subgoal (ID 383)
a : bool
l : list bool
H : l = [a]
=====
(hdb [a] && false)%bool = false

```

Below the goal, the ML4PG tool provides a similarity analysis:

```

-:%%- *goals*      All L7      (Coq Goals)
Similarities:
-----
6
This lemma is similar to the lemmas:
- M1_corrected- andb_false_r
-----
6
[]

```

The status bar at the bottom indicates the current file is 'ml4pg.v', the cursor is at line 1927, and the display mode is 'All L5 (Fundamental)'.

# A proof in Coq with ML4PG help

The screenshot shows a Coq IDE window titled 'ml4pg.v'. The left pane contains a Coq proof script, and the right pane shows an error message and a similarity search result.

```

end.

Lemma M1_corrected : forall l: list A, l = []
  -> tl (tl (tl l) ++ nil) = nil.
Proof.
intro l.
intro H.
rewrite H.
rewrite app_nil_l2.
simpl; trivial.
Qed.

Lemma andb_false_r : forall (a : bool) , false = andb a false.
Proof.
intros.
case a.
  simpl; trivial.
simpl; trivial.
Qed.

Lemma M3_3b : forall (a: bool) (l :list bool), l = [a] -> andb (hdb [a]) fals
e = false.
Proof.
intro l.
intro H.
rewrite H.
rewrite app_nil_l2.
simpl; trivial.
Qed.

```

The right pane displays an error message: "Error: Cannot find a relation to rewrite." Below this, a similarity search result is shown:

```

-:~*~ *response* All L1 (Coq Response)
Similarities:
-----
This lemma is similar to the lemmas:
- M1_corrected- andb_false_r
-----
[]

```

The status bar at the bottom indicates the file 'ml4pg.v', the current position 'Bot L929', the branch 'Git-master', the script '(Coq Script(1-) Holes)', the display mode 'All L5', and the level '(Fundamental)'. A message 'C-c 4 is undefined' is also visible.

# A proof in Coq with ML4PG help

The screenshot shows a Coq IDE window titled 'ml4pg.v'. The left pane contains the following Coq code:

```

end.

Lemma M1_corrected : forall l: list A, l = []
  -> tl (tl (tl l) ++ nil) = nil.
Proof.
intro l.
intro H.
rewrite H.
rewrite app_nil_l2.
simpl; trivial.
Qed.

Lemma andb_false_r : forall (a : bool) , false = andb a false.
Proof.
intros.
case a.
  simpl; trivial.
simpl; trivial.
Qed.

Lemma M3_3b : forall (a: bool) (l :list bool), l = [a] -> andb (hdb [a]) fals
e = false.
Proof.
intros.
case a.
  simpl; trivial.
simpl; trivial.
Qed.

```

The right pane shows the execution output:

```

M3_3b is defined

-:~%~ *response* All L1 (Coq Response)
Similarities:
-----
This lemma is similar to the lemmas:
- M1_corrected- andb_false_r
-----
[]

```

The status bar at the bottom shows: 'ml4pg.v Bot L931 Git-master (Coq Script(0-) Holes) -:~%~ \*display\* All L5 (Fundamental)'

# A proof in Coq with ML4PG help

The screenshot shows a Coq IDE window titled 'ml4pg.v'. The left pane contains the following Coq code:

```

end.

Lemma M1_corrected : forall l: list A, l = []
  -> tl (tl (tl l) ++ nil) = nil.
Proof.
intro l.
intro H.
rewrite H.
rewrite app_nil_l2.
simpl; trivial.
Qed.

Lemma andb_false_r : forall (a : bool) , false = andb a false.
Proof.
intros.
case a.
  simpl; trivial.
simpl; trivial.
Qed.

Lemma M3_3b : forall (a: bool) (l :list bool), l = [a] -> andb (hdb [a]) fals
e = false.
Proof.
intros.
rewrite <- andb_false_r.
trivial.
Qed.

```

The right pane shows the execution output:

```

M3_3b is defined

-:~%~ *response* All L1 (Coq Response)
Similarities:
-----
This lemma is similar to the lemmas:
- M1_corrected- andb_false_r
-----
[]

```

The status bar at the bottom shows: 'ml4pg.v Bot L930 Git-master (Coq Script(1-) Holes) -:~%~ \*display\* All L5 (Fundamental)'

# Research Problem

## Can clusters help with proof discovery?

Three methods have been created to automatically analogize proofs from these clusters.

These methods look to show that:

- Clusters created by ML4PG contain similar lemmas.
- New proofs can be analogized from these clusters that brute force would be unable to find.

# Simple Search

Method:

- For each lemma in cluster copy entire proof and see if it is valid in current lemma.

Example:

Prove lemma:

**Lemma** `plus_Sn_m` : `forall n m:nat, S n + m = S (n + m)`.

With Cluster: `aux7_bis`, `mulnS`, `mult_n_O`, `aux10`.

# Simple Search Example

```
Lemma aux7_bis : forall a:nat, a-a = 0.
```

```
Proof.
```

```
induction a.
```

```
  simpl; trivial.
```

```
simpl; trivial.
```

```
Qed.
```

```
Lemma plus_Sn_m : forall n m:nat, S n + m = S (n + m).
```

```
Proof.
```

```
induction a.
```

```
  simpl; trivial.
```

```
simpl; trivial.
```

```
Qed.
```

Error.

Searching mulnS, mult\_n\_0, aux10.

# Simple Search Example

**Lemma** mulnS : forall n m, n \* S m = n + n \* m.

**Proof.**

induction n.

trivial. intro m.

rewrite mulSn. rewrite mulSn. rewrite addSn. rewrite addSn. rewrite addnCA.

rewrite IHn. trivial.

**Qed.**

**Lemma** plus\_Sn\_m : forall n m:nat, S n + m = S (n + m).

induction n.

trivial. intro m.

rewrite mulSn. rewrite mulSn. rewrite addSn. rewrite addSn. rewrite addnCA.

rewrite IHn. trivial.

**Qed.**

**Error.**

Searching mult\_n\_0, aux10.



# Simple Search Example

```
Lemma mult_n_0 : forall n:nat, 0 = n * 0.
```

```
Proof.
```

```
induction n.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
Qed.
```

```
Lemma plus_Sn_m : forall n m:nat, S n + m = S (n + m).
```

```
Proof.
```

```
induction n.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
Qed.
```

Proof Solved.

# Simple Search

Success of simple search shows evidence towards the clusters being correct.

For Example:

Library	Size	Simple	SimpleBrute
Experimental	50	31 $\approx$ 62%	40 $\approx$ 80%
Paths (in CoQ HoTT library )	41	38 $\approx$ 93%	39 $\approx$ 95%

# Depth First Search

Method:

- 1 Create list of lists of all tactics used in proofs of other lemmas in clusters.
- 2 Depth first search the list of tactics until proof is found or no tactics remaining.

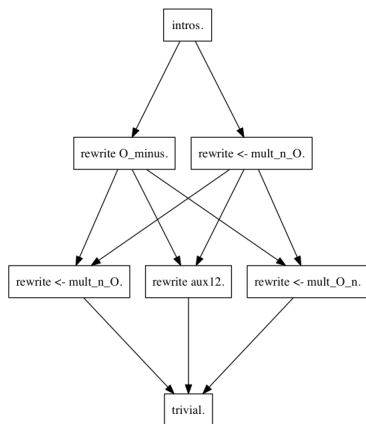
Example:

Prove lemma:

**Lemma** M26 : forall a b: nat, (0 - a) \* S b = 0.

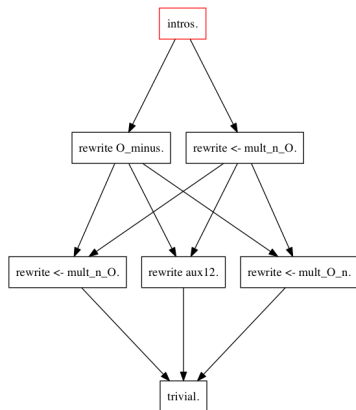
With Cluster: M41, M37, M32, M31, M22

# Depth First Search Proof Tree



# Depth First Search Example

Lemma M26 : forall a b: nat, (0 - a)  
 \* S b = 0.  
 Proof.  
 intros.



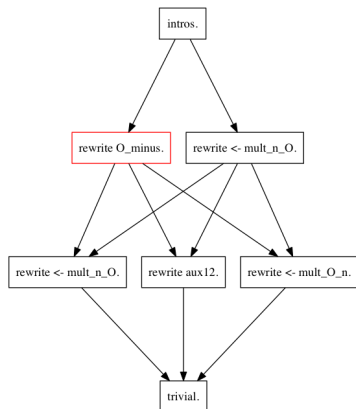
# Depth First Search Example

Lemma M26 : forall a b: nat, (0 - a)  
 \* S b = 0.

Proof.

intros.

rewrite 0\_minus.

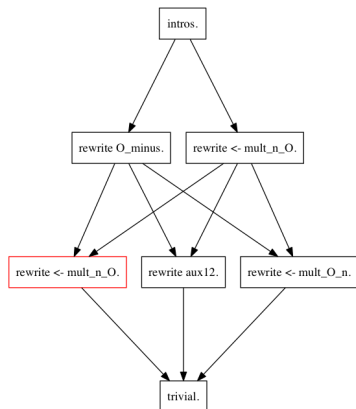


# Depth First Search Example

Lemma M26 : forall a b: nat, (0 - a)  
 \* S b = 0.

Proof.

```
intros.
rewrite 0_minus.
rewrite <- mult_n_0.
```



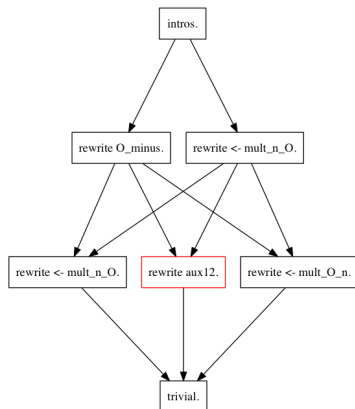
Error.

# Depth First Search Example

Lemma M26 : forall a b: nat, (0 - a)  
 \* S b = 0.

Proof.

```
intros.
rewrite 0_minus.
rewrite <- aux12.
```



Error.



# Depth First Search Example

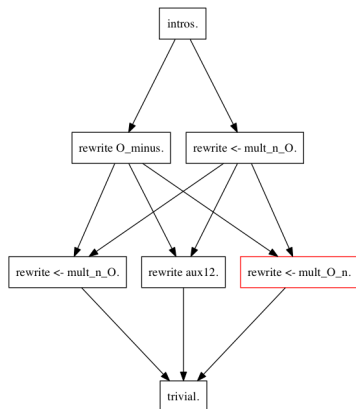
Lemma M26 : forall a b: nat, (0 - a)  
 \* S b = 0.

Proof.

intros.

rewrite 0\_minus.

rewrite <- mult\_0\_n.



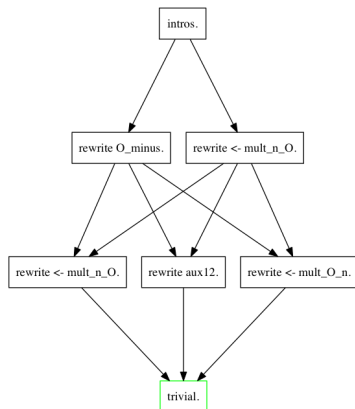
# Depth First Search Example

Lemma M26 : forall a b: nat, (0 - a)  
 \* S b = 0.

Proof.

```
intros.
rewrite O_minus.
rewrite <- mult_0_n.
trivial.
```

Qed.



Proof Solved.

# Context Mining Search

Method:

- 1 Extract each lemma removing internal variable references.
- 2 Perform a depth first search on the extracted lemmas using variables from the context instead of the internal ones.
- 3 If there is a reference to an external lemma all other lemmas in its cluster are also tried.

# Context Mining Search Example

Example:

Prove lemma:

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

With Cluster: andb\_false\_r, aux11, M1\_corrected, aux12, mulSn, addSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l, mulnS, aux7, addnCA, addnS

# Context Mining Search Example

How context mining search represents the proof found:

```
(1 . "induction")
(semi (0 . "simpl") (0 . "trivial"))
(semi (0 . "simpl") (0 . "trivial"))
(ext "rewrite" . "addSn")
(ext "rewrite" . "addnCA")
(1 . "rewrite")
(0 . "trivial")
```

# Context Mining Search Example

(1 . "induction")

One variable used in tactic. Possible variables from context: a

`Lemma M23 : forall a: nat, (a + 0) * S 0 = a.`

`Proof.`

`induction a.`

# Context Mining Search Example

```
(semi (0 . "simpl") (0 . "trivial"))
```

No variables used in tactics and tactics are separated by a semi colon.

```
Lemma M23 : forall a: nat, (a + 0) * S 0 = a.
```

```
Proof.
```

```
induction a.
```

```
simpl; trivial.
```

# Context Mining Search Example

```
(semi (0 . "simpl") (0 . "trivial"))
```

No variables used in tactics and tactics are separated by a semi colon.

```
Lemma M23 : forall a: nat, (a + 0) * S 0 = a.
```

```
Proof.
```

```
induction a.
```

```
simpl; trivial.
```

```
simpl; trivial.
```



# Context Mining Search Example

(ext "rewrite" . "addSn")

External rewrite with no arrows referenced.

Perform rewrite on variables in addSn clusters: addSn, andb\_false\_r, M23, aux11, M1\_corrected, aux12, mulSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite addSn.

**Error.**

# Context Mining Search Example

Remaining lemmas: `andb_false_r`, `M23`, `aux11`, `M1_corrected`, `aux12`, `mulSn`, `plus_0_n`, `app_nil_l2b`, `app_nil_l`

`Lemma M23 : forall a: nat, (a + 0) * S 0 = a.`

`Proof.`

`induction a.`

`simpl; trivial.`

`simpl; trivial.`

`rewrite andb_false_r.`

**Error.**

# Context Mining Search Example

Remaining lemmas: M23, aux11, M1\_corrected, aux12, mulSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

```
induction a.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
rewrite M23.
```

**Error.**

# Context Mining Search Example

Remaining lemmas: aux11, M1\_corrected, aux12, mulSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite aux11.

**Error.**

# Context Mining Search Example

Remaining lemmas: M1\_corrected, aux12, mulSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite M1\_corrected.

**Error.**

# Context Mining Search Example

Remaining lemmas: aux12, mulSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l

Lemma M23 : forall a: nat, (a + 0) \* S 0 = a.

Proof.

induction a.

simpl; trivial.

simpl; trivial.

rewrite aux12.

Error.

# Context Mining Search Example

Remaining lemmas: mulSn, plus\_0\_n, app\_nil\_l2b, app\_nil\_l

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite mulSn.

**Error.**

# Context Mining Search Example

Remaining lemmas: plus\_0\_n, app\_nil\_l2b, app\_nil\_l

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite plus\_0\_n.



# Context Mining Search Example

(ext "rewrite" . "addnS")

External rewrite with no arrows referenced. Perform rewrite on variables in addnCA, M23, mulnS, aux7, addnS

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite plus\_0\_n.

rewrite addnCA.

**Error.**

# Context Mining Search Example

Remaining lemmas: M23, mulnS, aux7, addnS

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

```
induction a.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
rewrite plus_0_n.
```

```
rewrite M23.
```

**Error.**

# Context Mining Search Example

Remaining lemmas: mulnS, aux7, addnS

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite plus\_0\_n.

rewrite mulnS.

**Error.**

# Context Mining Search Example

Remaining lemmas: aux7, addnS

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

```
induction a.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
rewrite plus_0_n.
```

```
rewrite aux7.
```

**Error.**

# Context Mining Search Example

Remaining lemmas: addnS

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

```
induction a.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
rewrite plus_0_n.
```

```
rewrite addnS.
```

# Context Mining Search Example

(1 . "rewrite")

Two variables available IHa and a. Trying rewrite on both.

**Lemma** M23 : forall a: nat, (a + 0) \* S 0 = a.

**Proof.**

induction a.

simpl; trivial.

simpl; trivial.

rewrite plus\_0\_n.

rewrite addnS.

rewrite IHa.

# Context Mining Search Example

```
(0 . "trivial")
```

No variables used in tactic

```
Lemma M23 : forall a: nat, (a + 0) * S 0 = a.
```

```
Proof.
```

```
induction a.
```

```
simpl; trivial.
```

```
simpl; trivial.
```

```
rewrite plus_0_n.
```

```
rewrite addnS.
```

```
rewrite IHa.
```

```
trivial.
```

```
Qed.
```

Proof Solved.

# Another Context Mining Search Example

## CompCert Proof

**Lemma** `ireg_of_eq` :

```
forall r r', ireg_of r = OK r' -> preg_of r = IR r'.
```

**Proof.**

```
unfold ireg_of; intros. destruct (preg_of r); inv H; auto.
```

**Qed.**



# Another Context Mining Search Example

## CompCert Proof

```
Lemma ireg_of_eq :  
  forall r r', ireg_of r = OK r' -> preg_of r = IR r'.  
Proof.  
  unfold ireg_of; intros. destruct (preg_of r); inv H; auto.  
Qed.
```

## Context Mining Proof

```
Lemma ireg_of_eq :  
  forall r r', ireg_of r = OK r' -> preg_of r = IR r'.  
Proof.  
  intros.  
  destruct r, r'; inv H; auto .  
Qed.
```

# Context Mining Advantages

- Makes use of clustering to find additional lemmas to rewrite and apply.
- Stops errors due to using incorrect variable name.
- Finds brand new proof which cannot be found by brute force.

# Method Results

This table only counts lemmas that are in a cluster.

Library	Size	Simple	DFS	CMS	Total
Experimental	50	$\approx 62\%$	$\approx 66\%$	$\approx 76\%$	$\approx 80\%$
Paths (in CoQ HoTT library )	41	$\approx 93\%$	$\approx 93\%$	$\approx 80\%$	$\approx 93\%$

Pending: CompCert

# Conclusion

- An add on for Proof General has been created for automatic analogizing of Coq Proofs.
- Three methods for analogizing Coq proofs from ML4PG clusters in proof general have been created.
- Clustering performed by ML4PG has been shown to find similar lemmas.
- More complex searching algorithms can be run on these clusters to find new proofs.

# Conclusion

- An add on for Proof General has been created for automatic analogizing of Coq Proofs.
- Three methods for analogizing Coq proofs from ML4PG clusters in proof general have been created.
- Clustering performed by ML4PG has been shown to find similar lemmas.
- More complex searching algorithms can be run on these clusters to find new proofs.

Further Work?