Statistical Machine Learning in Interactive Theorem Proving

Katya Komendantskaya and Jonathan Heras
(Funded by EPSRC First Grant Scheme)

University of Dundee

8 November 2013
Outline

1. Introduction

ML4PG: “Machine Learning for Proof General”

Using ML4PG

More Examples
- Detecting patterns across mathematical libraries
- Detecting irrelevant libraries

Conclusions and Further work
Outline

1. Introduction

2. ML4PG: “Machine Learning for Proof General”
Outline

1. Introduction

2. ML4PG: “Machine Learning for Proof General”

3. Using ML4PG

Detecting patterns across mathematical libraries
Detecting irrelevant libraries
Outline

1. Introduction
2. ML4PG: “Machine Learning for Proof General”
3. Using ML4PG
4. More Examples
   - Detecting patterns across mathematical libraries
   - Detecting irrelevant libraries
Outline

1. Introduction
2. ML4PG: “Machine Learning for Proof General”
3. Using ML4PG
4. More Examples
   - Detecting patterns across mathematical libraries
   - Detecting irrelevant libraries
5. Conclusions and Further work
Interactive theorem proving:

- (typically) higher-order language (Agda, Coq, Isabelle/HOL)
- (often) dependently-typed (AGDA, Coq)
- Interactive proof development: tactic – prover response;
- Expressive enough to verify large areas of Maths, software, hardware.
Interactive theorem proving:

- (typically) higher-order language (Agda, Coq, Isabelle/HOL)
- (often) dependently-typed (AGDA, Coq)
- Interactive proof development: tactic – prover response;
- Expressive enough to verify large areas of Maths, software, hardware.
  - ...enriched with dependent types, (co)inductive types, type classes and provide rich programming environments;
  - ...applied in formal mathematical proofs: Four Colour Theorem (60,000 lines), Kepler conjecture (325,000 lines), Feit-Thompson Theorem (170,000 lines), etc.
  - ...applied in industrial proofs: seL4 microkernel (200,000 lines), verified C compiler (50,000 lines), ARM microprocessor (20,000 lines), etc.
Coq and SSReflect

- SSReflect is a dialect of Coq;
- The SSReflect library was developed as the infrastructure for formalisation of the Four Colour Theorem;
- played a key role in the formal proof of the Feit-Thompson theorem.


Introduction

Challenges

- ...size and sophistication of libraries stand on the way of efficient knowledge reuse;
- ...manual handling of various proofs, strategies, libraries, becomes difficult;
- ...team-development is hard, especially that ITPs are sensitive to notation;
- ...comparison of proof similarities is hard.
Introduction

An example: JVM

Java Virtual Machine (JVM) is a stack-based abstract machine which can execute Java bytecode.
An example: JVM

Java Virtual Machine (JVM) is a stack-based abstract machine which can execute Java bytecode.

Goal

- Model a subset of the JVM in (e.g.) CoQ, defining an interpreter for JVM programs,
- Verify the correctness of JVM programs within CoQ.
Introduction

An example: JVM

Java Virtual Machine (JVM) is a stack-based abstract machine which can execute Java bytecode.

Goal

- Model a subset of the JVM in (e.g.) CoQ, defining an interpreter for JVM programs,
- Verify the correctness of JVM programs within CoQ.

This work is inspired by:

Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter:

0

stack:

[ ] [ ] [ ] ...

local variables:

5 [ ] [ ] ...

Goal (Factorial case)

∀ n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>iconst 1</code></td>
</tr>
<tr>
<td>1</td>
<td><code>istore 1</code></td>
</tr>
<tr>
<td>2</td>
<td><code>iload 0</code></td>
</tr>
<tr>
<td>3</td>
<td><code>ifeq 13</code></td>
</tr>
<tr>
<td>4</td>
<td><code>iload 1</code></td>
</tr>
<tr>
<td>5</td>
<td><code>iload 0</code></td>
</tr>
<tr>
<td>6</td>
<td><code>imul</code></td>
</tr>
<tr>
<td>7</td>
<td><code>istore 1</code></td>
</tr>
<tr>
<td>8</td>
<td><code>iload 0</code></td>
</tr>
<tr>
<td>9</td>
<td><code>iconst 1</code></td>
</tr>
<tr>
<td>10</td>
<td><code>isub</code></td>
</tr>
<tr>
<td>11</td>
<td><code>istore 0</code></td>
</tr>
<tr>
<td>12</td>
<td><code>goto 2</code></td>
</tr>
<tr>
<td>13</td>
<td><code>iload 1</code></td>
</tr>
<tr>
<td>14</td>
<td><code>ireturn</code></td>
</tr>
</tbody>
</table>

JVM model:

- **counter:** 1
- **stack:**
  - 1
  - ...
- **local variables:**
  - 5
  - ...

Goal (Factorial case)

∀ \( n \in \mathbb{N} \), running the bytecode associated with the factorial program with \( n \) as input produces a state which contains \( n! \) on top of the stack.
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0  :  iconst 1
1  :  istore 1
2  :  iload 0
3  :  ifeq 13
4  :  iload 1
5  :  iload 0
6  :  imul
7  :  istore 1
8  :  iload 0
9  :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter: 2

stack:...

local variables:

5 1 ...

Goal (Factorial case)

∀n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  icnst 1
10 : isub
11 : istore 0
12 : goto 2
13 : iload 1
14 : ireturn
```

JVM model:

counter: 3

stack: 5 ... 

local variables: 5 1 ...
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter:
4

stack:

local variables:

```
5 1 ...
```

Goal (Factorial case)

∀ n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter:

5

stack:

```
1
```

local variables:

```
5 1 ...
```
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 : isub
11 : istore 0
12 : goto 2
13 : iload 1
14 : ireturn
```

JVM model:

counter: 6

stack: 5 1 ...

local variables: 5 1 ...

Goal (Factorial case)

∀ n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

```
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 : isub
11 : istore 0
12 : goto 2
13 : iload 1
14 : ireturn
```

JVM model:

counter:

7

stack:

```
5 5 ... 5 1 ...
```

local variables:

```
5 5 ...
```

Goal (Factorial case)

\( \forall n \in \mathbb{N}, \text{running the bytecode associated with the factorial program with } n \text{ as input produces a state which contains } n! \text{ on top of the stack.} \)
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter:

8

stack:

```
      . . .
```

local variables:

```
5 5 . . .
```
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```
0 : iconst 1
1 : istore 1
2 : iload 0
3 : ifeq 13
4 : iload 1
5 : iload 0
6 : imul
7 : istore 1
8 : iload 0
9 : iconst 1
10 : isub
11 : istore 0
12 : goto 2
13 : iload 1
14 : ireturn
```

JVM model:

counter:
9

stack:

```
5
```

local variables:

```
5
5
```

Goal (Factorial case)

∀n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}

Bytecode:

JVM model:

Goal (Factorial case)

∀ n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter: 11

stack: 4 ...

local variables: 5 5 ...

Goal (Factorial case)

∀ n ∈ N, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  icnst 1
10 : isub
11 : istore 0
12 : goto 2
13 : iload 1
14 : ireturn
```

JVM model:

- **counter:** 12
- **stack:** ...
- **local variables:** 
  ```
  4 5 ...
  ```
Computing 5!

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter:

2

stack:

[...]

local variables:

[4 5 [...]]
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```none
...  
```

JVM model:

```none
...  
```
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

counter: 13

stack:

```
[0] [ ] [ ] [...]
```

local variables:

```
[0] [120] [ ] [...]
```
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```plaintext
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

- **counter:** 14
- **stack:**
  - 120 [ ... ]
- **local variables:**
  - 0 120 [ ... ]
Computing 5! 

Java code:

```java
static int factorial(int n)
{
    int a = 1;
    while (n != 0){
        a = a * n;
        n = n-1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  iconst 1
10 :  isub
11 :  istore 0
12 :  goto 2
13 :  iload 1
14 :  ireturn
```

JVM model:

- counter: 15
- stack: [120, ...]
- local variables: [0, 120, ...]
Computing 5!

Java code:

```java
static int factorial(int n) {
    int a = 1;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

Bytecode:

```
0 :  iconst 1
1 :  istore 1
2 :  iload 0
3 :  ifeq 13
4 :  iload 1
5 :  iload 0
6 :  imul
7 :  istore 1
8 :  iload 0
9 :  icnst 1
10 : isub
11 : istore 0
12 : goto 2
13 : iload 1
14 : ireturn
```

JVM model:

- counter: 15
- stack:
  - 120 ...  
- local variables:
  - 0 120 ...  

Goal (Factorial case)

\[ \forall n \in \mathbb{N}, \text{ running the bytecode associated with the factorial program with } n \text{ as input produces a state which contains } n! \text{ on top of the stack.} \]
Goal (Factorial case)

∀ n ∈ ℕ, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.
Formalisation of Java bytecode in Coq

Goal (Factorial case)

\( \forall n \in \mathbb{N}, \) running the bytecode associated with the factorial program with \( n \) as input produces a state which contains \( n! \) on top of the stack.

Methodology:

1. Write the specification of the function

\[
\text{Definition theta_fact} \ (n : \text{nat}) := n'!.
\]
Introduction

Formalisation of Java bytecode in Coq

Goal (Factorial case)

\[ \forall n \in \mathbb{N}, \text{running the bytecode associated with the factorial program with } n \text{ as input produces a state which contains } n! \text{ on top of the stack.} \]

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)

\[
\text{Fixpoint helper_fact (n a : nat) :=}
\begin{align*}
\text{match n with} \\
\text{| 0 => a} \\
\text{| S p => helper_fact p (n * a)} \\
\text{end.}
\end{align*}
\]

\[
\text{Definition fn_fact (n : nat) := helper_fact n 1.}
\]
Formalisation of Java bytecode in Coq

Goal (Factorial case)

∀ \( n \in \mathbb{N} \), running the bytecode associated with the factorial program with \( n \) as input produces a state which contains \( n! \) on top of the stack.

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification

Lemma fn_fact_is_theta n :
fn_fact n = theta_fact n.
Formalisation of Java bytecode in Coq

Goal (Factorial case)
∀ n ∈ ℤ, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification
4. Write the JVM program

Definition pi_fact :=
[::(ICONST,1%Z);
 (ISTORE,1%Z);
 (ILOAD,0%Z);
 (IFEQ,10%Z);
 (ILOAD,1%Z);
 (ILOAD,0%Z);
 (IMUL, 0%Z);
 (ISTORE, 1%Z);
 (ILOAD, 0%Z);
 (ICONST, 1%Z);
 (ISUB, 0%Z);
 (ISTORE, 0%Z);
 (GOTO, (-10)%Z);
 (ILOAD, 1%Z);
 (HALT, 0%Z)].
Formalisation of Java bytecode in Coq

Goal (Factorial case)

\( \forall n \in \mathbb{N}, \) running the bytecode associated with the factorial program with \( n \) as input produces a state which contains \( n! \) on top of the stack.

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification
4. Write the JVM program
5. Define the function that schedules the program

\[\begin{align*}
\text{Fixpoint } & \text{loop_sched_fact (n : nat) := } \\
& \quad \text{match n with } \\
& \quad | 0 \Rightarrow \text{nseq 3 0} \\
& \quad | S p \Rightarrow \text{nseq 11 0 ++ loop_sched_fact p} \\
& \quad \text{end.} \\
\end{align*}\]

\[\begin{align*}
\text{Definition } & \text{sched_fact (n : nat) := } \\
& \quad \text{nseq 2 0 ++ loop_sched_fact n.} \\
\end{align*}\]
Formalisation of Java bytecode in Coq

Goal (Factorial case)
∀ n ∈ ℕ, running the bytecode associated with the factorial program with n as input produces a state which contains n! on top of the stack.

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification
4. Write the JVM program
5. Define the function that schedules the program
6. Prove that the code implements the algorithm

Lemma program_is_fn_fact n :
run (sched_fact n)
  (make_state 0 [::n] [::] pi_fact) =
  (make_state 14 [::0;fn_fact n ]
    (push (fn_fact n ) [::]) pi_fact).
Formalisation of Java bytecode in Coq

Goal (Factorial case)

\( \forall n \in \mathbb{N}, \) running the bytecode associated with the factorial program with \( n \) as input produces a state which contains \( n! \) on top of the stack.

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification
4. Write the JVM program
5. Define the function that schedules the program
6. Prove that the code implements the algorithm
7. Prove total correctness

\textbf{Theorem} total_correctness_fact n sf :

\[ sf = \text{run (sched_fact n)} \]
\[ (\text{make_state} 0 [::n] [::] \text{pi_fact}) \rightarrow \]
\[ \text{next_inst sf} = (\text{HALT},0\%Z) \land \]
\[ \text{top (stack sf)} = (n)! \]
Formalisation of Java bytecode in Coq

Goal (Factorial case)

∀ \( n \in \mathbb{N} \), running the bytecode associated with the factorial program with \( n \) as input produces a state which contains \( n! \) on top of the stack.

Methodology:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification
4. Write the JVM program
5. Define the function that schedules the program
6. Prove that the code implements the algorithm
7. Prove total correctness

Lemma fn_fact_is_theta n :
fn_fact n = theta_fact n.
Proof of lemma \texttt{fn\_fact\_is\_theta}

Le lemma \texttt{fn\_fact\_is\_theta} : \texttt{forall (n : nat), fn\_fact n = theta\_fact n}.

Proof.

1 subgoals, subgoal 1 (ID 13)

\begin{verbatim}
    Proof.
    1. \forall n : \text{nat}, \text{fn\_fact n} = \text{theta\_fact n}
\end{verbatim}
**Proof of lemma** \( fn\_fact\_is\_theta \)

Lemma \( fn\_fact\_is\_theta : \forall n : \text{nat}, fn\_fact n = \theta\_fact n. \)

Proof.

```coq
move => n.
```

1 subgoals, subgoal 1 (ID 14)

\( n : \text{nat} \)

```coq
= fn\_fact n = \theta\_fact n
```

Katya and Jónathan (Dundee)
Proof of lemma \texttt{fn\_fact\_is\_theta}

\textbf{Lemma} \texttt{fn\_fact\_is\_theta} : \texttt{forall (n : nat), fn\_fact n = theta\_fact n}.

\textbf{Proof}.
\begin{itemize}
  \item move => n.
  \item rewrite /fn\_fact /theta\_fact.
\end{itemize}

1 subgoals, subgoal 1 (ID 14)

\begin{itemize}
  \item n : nat
  \item \texttt{helper\_fact n 1 = n'}!
\end{itemize}
Proof of lemma fn_fact_is_theta

Lemma fn_fact_is_theta : forall (n : nat), fn_fact n = theta_fact n.
Proof.
move => n.
rewrite /fn_fact /theta_fact.

1 subgoals, subgoal 1 (ID 14)

n : nat
============================
helper_fact n 1 = n'!

...and now?
Outline

1. Introduction
2. ML4PG: “Machine Learning for Proof General”
3. Using ML4PG
4. More Examples
   - Detecting patterns across mathematical libraries
   - Detecting irrelevant libraries
5. Conclusions and Further work
Machine Learning 4 Proof General: interfacing interfaces

...in [2013, Postproc. of UITP’12]

Proof General

Interactive Prover: Coq, SSReflect

ML4PG

feature extraction

proof families

MATLAB/Weka

Clustering:
K-means, Gaussian, ...
Machine Learning 4 Proof General: interfacing interfaces

...in [2013, Postproc. of UITP’12]

F.1. works on the background of Proof General extracting some low-level features from proofs in Coq/SSReflect.

F.2. automatically sends the gathered statistics to a chosen machine-learning interface and triggers execution of a clustering algorithm of user’s choice;

F.3. does some post-processing of the results and displays families of related proofs to the user.
Features of this approach

Feature extraction:
- features are extracted from higher-order propositions and proofs;
- feature extraction is built on the method of proof-traces;
- longer proofs are analysed by means of the proof-patch method.

Diagram:

- **Proof General**
  - Interactive Prover: Coq, SSReflect

- **ML4PG**
  - proof families

- **MATLAB/Weka**
  - Clustering: K-means, Gaussian, ...
What are the significant features of proofs?

1-2 names and the number of tactics used in one command line,

3 types of the tactic arguments;

4 relation of the tactic arguments to the (inductive) hypotheses or library lemmas,

5-7 three top symbols in the term-tree of the current subgoal, and

8 the number of subgoals each tactic command-line generates.
What are the significant features of proofs?

1-2 names and the number of tactics used in one command line,
3 types of the tactic arguments;
4 relation of the tactic arguments to the (inductive) hypotheses or library lemmas,
5-7 three top symbols in the term-tree of the current subgoal, and
8 the number of subgoals each tactic command-line generates.

Taken within 5 proof steps;
...40 features for one proof patch.
Thus a proof fragment is given by a point in a 40-dimensional space.
Features of this approach

Machine-learning tools:
- works with unsupervised learning (clustering) algorithms implemented in MATLAB and Weka;
- uses algorithms such as Gaussian, K-means, and farthest-first.
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:

  ![Diagram of data points grouped into clusters]

- 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, …
Order your own copy of ML4PG!

- ML4PG is now a part of standard Proof General distribution
- Easy to find: just google “ML4PG” for our page with all software resources, libraries of examples, papers, etc.

This talk:

J. Heras and K. Komendantskaya. Recycling Proof-Patterns in Coq: Case Studies. 31 page. Submitted, available in ARXIV.

- Easy to install;
Order your own copy of ML4PG!

- ML4PG is now a part of standard Proof General distribution
- Easy to find: just google “ML4PG” for our page with all software resources, libraries of examples, papers, etc.

This talk:


- Easy to install;
- Easy to use?
Outline

1. Introduction
2. ML4PG: “Machine Learning for Proof General”
3. Using ML4PG
4. More Examples
5. Conclusions and Further work
Continuation of proof of lemma \texttt{fn\_fact\_is\_theta}

\begin{verbatim}
Lemma fn_fact_is_theta : forall (n : nat), fn_fact n = theta_fact n.
Proof.
move => n.
rewrite /fn_fact /theta_fact.

1 subgoals, subgoal 1 (ID 14)

n : nat
============================
helper_fact n 1 = n'!
\end{verbatim}
Continuation of proof of lemma `fn_fact_is_theta`

Lemma `fn_fact_is_theta` : `forall (n : nat), fn_fact n = theta_fact n`.

Proof.
move => n.
rewrite /fn_fact /theta_fact.

1 subgoals, subgoal 1 (ID 14)

n : nat

`helper_fact n 1 = n'`!
Continuation of proof of lemma \texttt{fn\_fact\_is\_theta}

Lemma \texttt{fn\_fact\_is\_theta} : \texttt{forall} (n : \texttt{nat}), \texttt{fn\_fact} n = \texttt{theta\_fact} n.

Proof.
move => n.
rewrite /\texttt{fn\_fact} /\texttt{theta\_fact}.

1 subgoals, subgoal 1 (ID 14)

\texttt{n : nat}
\texttt{helper\_fact} n 1 = n'!

\texttt{Lemma fn\_fact\_is\_theta is similar to lemmas:}
\texttt{- fn\_expt\_is\_theta}
\texttt{- fn\_mult\_is\_theta}
\texttt{- fn\_power\_is\_theta}
# Proving lemma `fn_fact_is_theta` by analogy

## Factorial

**Lemma** `fn_fact_is_theta n : fn_fact n = n`!.

**Proof.**

move => n. rewrite /fn_fact.

## Exponentiation

**Lemma** `fn_expt_is_theta n m : fn_expt n m = n^m`.

**Proof.**

by move => n; rewrite /fn_expt helper_expt_is_theta mul1n.

Qed.

**Lemma** `helper_expt_is_theta n m a :` 

`helper_expt n m a = a * (n ^ m)`.

**Proof.**

move : a; elim : n => [a| n IH a /=].

by rewrite /theta_expt expn0 muln1.

by rewrite IH /theta_expt expnS mulnA [a * _]mulnC.

Qed.
### Proving lemma \texttt{fn\_fact\_is\_theta} by analogy

<table>
<thead>
<tr>
<th>Factorial</th>
<th>Exponentiation</th>
</tr>
</thead>
</table>
| **Lemma** \texttt{fn\_fact\_is\_theta} \( n \) : \( \text{fn\_fact} \ n = n'! \).  
Proof.  
move => n. rewrite /fn\_fact. | **Lemma** \texttt{fn\_expt\_is\_theta} \( n \) \( m \) : \( \text{fn\_expt} \ n \ m = n^m \).  
Proof.  
by move => n; rewrite /fn\_expt helper\_expt\_is\_theta mul1n.  
Qed. |
| \texttt{Lemma} helper\_fact\_is\_theta \( n \) \( a \) :  
helper\_fact \( n \) \( a \) = \( a \) * \( n'! \).  
Proof.  
move : \( n \) \( a \); elim : \( m \) => \([a \ m] \ m \) IH \( n \) \( a \) /=.  
by rewrite /theta\_fact fact0 muln1.  
by rewrite IH /theta\_fact factS  
mulnA \([a \ * \ ]\)mulnC.  
Qed. | \texttt{Lemma} helper\_expt\_is\_theta \( n \) \( m \) \( a \) :  
helper\_expt \( n \) \( m \) \( a \) = \( a \) * \( (n^m) \).  
Proof.  
move : \( a \); elim : \( n \) => \([a \ n] \ n \) IH \( a \) /=.  
by rewrite /theta\_expt expn0 muln1.  
by rewrite IH /theta\_expt expnS  
mulnA \([a \ * \ ]\)mulnC.  
Qed. |
### Proving lemma `fn_fact_is_theta` by analogy

#### Factorial

<table>
<thead>
<tr>
<th>Lemma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fn_fact_is_theta n : fn_fact n = n'!</code>.</td>
<td></td>
</tr>
<tr>
<td>Proof.</td>
<td></td>
</tr>
<tr>
<td>move =&gt; n. rewrite /fn_fact.</td>
<td></td>
</tr>
<tr>
<td>by rewrite helper_fact_is_theta mul1n.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lemma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>helper_fact_is_theta n a : helper_fact n a = a * n'!</code>.</td>
<td></td>
</tr>
<tr>
<td>Proof.</td>
<td></td>
</tr>
<tr>
<td>move : n a; elim : m =&gt; [a m</td>
<td>m IH n a /=].</td>
</tr>
<tr>
<td>by rewrite /theta_fact fact0 muln1.</td>
<td></td>
</tr>
<tr>
<td>by rewrite IH /theta_fact factS</td>
<td></td>
</tr>
<tr>
<td>mulnA [a * _]mulnC.</td>
<td></td>
</tr>
</tbody>
</table>

**Qed.**

#### Exponentiation

<table>
<thead>
<tr>
<th>Lemma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fn_expt_is_theta n m : fn_expt n m = n^m</code>.</td>
<td></td>
</tr>
<tr>
<td>Proof.</td>
<td></td>
</tr>
<tr>
<td>by move =&gt; n; rewrite /fn_expt helper_expt_is_theta mul1n.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lemma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>helper_expt_is_theta n m a : helper_expt n m a = a * (n ^ m)</code>.</td>
<td></td>
</tr>
<tr>
<td>Proof.</td>
<td></td>
</tr>
<tr>
<td>move : a; elim : n =&gt; [a</td>
<td>n IH a /=].</td>
</tr>
<tr>
<td>by rewrite /theta_expt expn0 muln1.</td>
<td></td>
</tr>
<tr>
<td>by rewrite IH /theta_expt expnS</td>
<td></td>
</tr>
<tr>
<td>mulnA [a * _]mulnC.</td>
<td></td>
</tr>
</tbody>
</table>

**Qed.**
# Proving lemma fn_fact_is_theta by analogy

<table>
<thead>
<tr>
<th>Factorial</th>
<th>Exponentiation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lemma</strong> fn_fact_is_theta n : fn_fact n = n'!.</td>
<td></td>
</tr>
<tr>
<td><strong>Proof.</strong></td>
<td></td>
</tr>
<tr>
<td>move =&gt; n. rewrite /fn_fact.</td>
<td></td>
</tr>
<tr>
<td>by rewrite helper_fact_is_theta mul1n.</td>
<td></td>
</tr>
<tr>
<td>Qed.</td>
<td></td>
</tr>
</tbody>
</table>

| **Lemma** helper_fact_is_theta n a : |
| helper_fact n a = a * n'!. |
| **Proof.** |
| move : n a; elim : m => [a m/ m IH n a /=]. |
| by rewrite /theta_fact fact0 muln1. |
| by rewrite IH /theta_fact factS |
| mulnA [a * _]mulnC. |
| Qed. |

| **Lemma** fn_expt_is_theta n m : fn_expt n m = n^m. |
| **Proof.** |
| by move => n; rewrite /fn_expt helper_expt_is_theta |
| mul1n. |
| Qed. |

| **Lemma** helper_expt_is_theta n m a : |
| helper_expt n m a = a * (n ^ m). |
| **Proof.** |
| move : a; elim : n => [a/ n IH a /=]. |
| by rewrite /theta_expt expn0 muln1. |
| by rewrite IH /theta_expt expnS |
| mulnA [a * _]mulnC. |
| Qed. |

## Proof Strategy

_Prove an auxiliary lemma about the helper considering the most general case. For example, if the helper function is defined with formal parameters n, m, and a, and the wrapper calls the helper initializing a at 0, the helper theorem must be about (helper n m a), not just about the special case (helper n m 0). Subsequently, instantiate the lemma for the concrete case._
### Consistency of ML4PG clusters

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( g = 1 ) ((n = 16))</th>
<th>( g = 2 ) ((n = 18))</th>
<th>( g = 3 ) ((n = 21))</th>
<th>( g = 4 ) ((n = 24))</th>
<th>( g = 5 ) ((n = 29))</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-means</td>
<td>( 30^{a,b,d} )</td>
<td>( 4^{a-d} )</td>
<td>( 4^{a-d} )</td>
<td>( 2^{c,d} )</td>
<td>0</td>
</tr>
<tr>
<td>E.M.</td>
<td>( 21^{a-d} )</td>
<td>( 7^{a-d} )</td>
<td>( 7^{a-d} )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FarthestFirst</td>
<td>( 28^{a-d} )</td>
<td>( 25^{a-d} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**a)** Lemma about JVM multiplication program  
**b)** Lemma about JVM power program  
**c)** Lemma about JVM exponentiation program  
**d)** Lemma about JVM factorial
Where else ML4PG can be applied?

Similarly, ML4PG can be used in:

1. Write the specification of the function
2. Write the algorithm (tail recursive function)
3. Prove that the algorithm satisfies the specification
4. Write the JVM program
5. Define the function that schedules the program
6. Prove that the code implements the algorithm
7. Prove total correctness
Proving lemma \texttt{program\_is\_fn\_fact} by analogy

Factorial

**Lemma** \texttt{program\_is\_fn\_fact} \(n\):

\[
\text{run (sched\_fact} \ n) (\text{make\_state} \ 0 \ [::n] \ [::] \ \text{pi\_fact}) = \\
(\text{make\_state} \ 14 \ [::0;fn\_fact} \ n \ ] \ (\text{push} \ (fn\_fact} \ n \ [::]) \ \text{pi\_fact}).
\]

**Proof.**

\texttt{rewrite run\_app.}
**Proving lemma** `program_is_fn_fact` **by analogy**

### Exponentiation (ML4PG suggestion)

<table>
<thead>
<tr>
<th>Lemma</th>
<th>program_is_fn_expt n m :</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>run (sched_expt n m)(make_state 0 [:n;m] [:] pi_expt)=</td>
</tr>
<tr>
<td></td>
<td>(make_state 14 [::0;fn_expt n m] (push (fn_expt n m)[:])pi_expt).</td>
</tr>
<tr>
<td>Proof.</td>
<td>rewrite run_app loop_is_helper_expt.</td>
</tr>
<tr>
<td></td>
<td>Qed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lemma</th>
<th>loop_is_helper_expt n m a :</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>run (loop_sched_expt n)(make_state 2 [:n;m;a] [:] pi_expt)=</td>
</tr>
<tr>
<td></td>
<td>(make_state 14 [::0;(helper_expt n m a)] (push (helper_expt n m a)[:])pi_expt)</td>
</tr>
<tr>
<td>Proof.</td>
<td>move : n a; elim : m =&gt; [/[//</td>
</tr>
<tr>
<td></td>
<td>by rewrite -IH subn1 -pred_Sn.</td>
</tr>
<tr>
<td></td>
<td>Qed.</td>
</tr>
</tbody>
</table>
Proving lemma `program_is_fn_fact` by analogy

---

**Factorial**

Lemma `program_is_fn_fact` n :

run (sched_fact n)(make_state 0 [:n] [:] pi_fact)=
  (make_state 14 [:0;fn_fact n ] (push (fn_fact n )[:])pi_fact).

Proof.

rewrite run_app.

Lemma `loop_is_helper_fact` n a :

run (loop_sched_fact n)(make_state 2 [:n;a] [:] pi_fact)=
  (make_state 14 [:0;(helper_fact n a)] (push (helper_fact n a)[:])pi_fact)

Proof.

move : a; elim : n => [// | n IH a].

by rewrite -IH subn1 -pred_Sn [- _ a]mulnC.

Qed.
Proving lemma `program_is_fn_fact` by analogy

---

**Factorial**

Lemma `program_is_fn_fact n`:

run (sched_fact n)(make_state 0 [:n] [:] pi_fact)=

(make_state 14 [:0;fn_fact n ] (push (fn_fact n )[:])pi_fact).

Proof.

rewrite run_app.

rewrite loop_is_helper_fact.

Qed.
Proving lemma \texttt{program\_is\_fn\_fact} by analogy

**Factorial**

\begin{verbatim}
Lemma program_is_fn_fact n :
  run (sched_fact n) (make_state 0 [:n] [:] pi_fact) =
  (make_state 14 [:0;fn_fact n] (push (fn_fact n) [:]) pi_fact).
Proof.
rewrite run_app.
rewrite loop_is_helper_fact.
Qed.
\end{verbatim}

**Proof Strategy**

*Prove that the loop implements the helper using an auxiliary lemma. Such a lemma about the loop must consider the general case as in the previous proof strategy. Subsequently, instantiate the result to the concrete case.*
Using ML4PG

Proving lemma `program_is_fn_fact` by analogy

**Factorial**

**Lemma** `program_is_fn_fact n` :

\[
\text{run} \left( \text{sched_fact} \ n \right) \left( \text{make_state} \ 0 \ \left[::n\right] \ \left[::\right] \ \text{pi_fact} \right) = \\
\left( \text{make_state} \ 14 \ \left[::0;\text{fn_fact} \ n\right] \ \left( \text{push} \ \left( \text{fn_fact} \ n \right) \left[::\right] \right) \ \text{pi_fact} \right).
\]

**Proof.**

`rewrite` `run_app`.  
`rewrite` `loop_is_helper_fact`.  
`Qed.`

**Proof Strategy**

*Prove that the loop implements the helper using an auxiliary lemma. Such a lemma about the loop must consider the general case as in the previous proof strategy. Subsequently, instantiate the result to the concrete case.*

ML4PG suggestions (for several parameters): Analogous theorems for multiplication, exponentiation and power.
Proving total correctness by analogy

Factorial

Theorem total_correctness_fact n sf :

\[ sf = \text{run (sched_fact n)(make_state 0 [::n] [::] pi_fact)->next_inst sf = (HALT,0\%Z)/\ top (stack sf)= (n')}. \]

Proof.
move => H; split
Using ML4PG

Proving total correctness by analogy

---

**Exponentiation (ML4PG suggestion)**

```coq
Theorem total_correctness_expt n m sf :
  sf = run (sched_expt m)(make_state 0 [::n;m] [::] pi_expt)->
  next_inst sf = (HALT,0%Z)/\ top (stack sf) = (n^m).

Proof.
  by move => H; split; rewrite H program_is_fn_expt fn_expt_is_theta.
Qed.
```
Proving total correctness by analogy

Factorial

**Theorem** total_correctness_fact n sf :

\[
sf = \text{run} (\text{sched_fact} n)(\text{make_state} \ 0 \ [::n] \ [::] \ \text{pi_fact}) \rightarrow \text{next_inst} \ sf = (\text{HALT}, 0 \% Z) \land \text{top}(\text{stack} \ sf) = (n')\]

**Proof.**

move \Rightarrow H; split

\[; \text{rewrite} \ H \ \text{program_is_fn_fact} \ \text{fn_fact_is_theta}.\]

Qed.
Proving total correctness by analogy

Factorial

\[\text{Theorem} \ \text{total\_correctness\_fact} \ n \ \text{sf} : \]
\[\text{sf} = \text{run} (\text{sched\_fact} \ n) (\text{make\_state} \ 0 \ [\vdots n] \ [\vdots] \text{pi\_fact}) \rightarrow \]
\[\text{next\_inst} \ \text{sf} = (\text{HALT}, 0\%Z) / \ \text{top} (\text{stack} \ \text{sf}) = (n'!).\]

\[\text{Proof.}\]
\[\text{move} \Rightarrow \ H; \ \text{split}\]
\[\ ; \ \text{rewrite} \ H \ \text{program\_is\_fn\_fact} \ \text{fn\_fact\_is\_theta}.\]

\[\text{Qed.}\]

Proof Strategy

*Combine lemmas of the two previous steps.*
Using ML4PG

Proving total correctness by analogy

Factorial

Theorem total_correctness_fact n sf :
\[ sf = \text{run} (\text{sched\_fact} n)(\text{make\_state} 0 :: n :: \text{pi\_fact}) \to \]
\[ \text{next\_inst} sf = (\text{HALT}, 0 \% Z) / \text{top} (\text{stack} sf) = (n!). \]

Proof.
move => H; split

; rewrite H program_is_fn_fact fn_fact_is_theta.
Qed.

Proof Strategy

*Combine lemmas of the two previous steps.*

ML4PG suggestions (for several parameters): Analogous theorems for multiplication, exponentiation and power.
Outline

1. Introduction

2. ML4PG: “Machine Learning for Proof General”

3. Using ML4PG

4. More Examples
   - Detecting patterns across mathematical libraries
   - Detecting irrelevant libraries

5. Conclusions and Further work
The bigop library

- **SSReflect** library about indexed big “operations”
The bigop library

- **SSReflect** library about indexed big “operations”
- Examples:
  \[
  \sum_{0 \leq i < 2n \text{ odd } i} i = n^2, \quad \prod_{0 \leq i \leq n} i = n!, \quad \bigcup_{i \in I} f(i), \ldots
  \]
The bigop library

- **SSReflect** library about indexed big “operations”
- Examples:
  \[
  \sum_{0 \leq i < 2n \text{ odd}} i = n^2, \quad \prod_{0 \leq i \leq n} i = n!, \quad \bigcup_{i \in I} f(i), \ldots
  \]
- Applications:
  - Definition of matrix multiplication
  - Binomials
  - Union of sets
  - ...
Application of ML4PG: Inverse of nilpotent matrices

**Definition**

Let $M$ be a square matrix, $M$ is nilpotent if it exists an $n$ such that $M^n = 0$.

\[
(1 - M) \times \sum_{0 \leq i < n} M^i = 1
\]
Application of ML4PG: Inverse of nilpotent matrices

**Definition**
Let $M$ be a square matrix, $M$ is nilpotent if it exists an $n$ such that $M^n = 0$.

**Lemma**
Let $M$ be a nilpotent matrix, then

$$(1 - M) \times \sum_{0 \leq i < n} M^i = 1$$

where $n$ is such that $M^n = 0$.

Lemma inverse_I_minus_M_big $\text{inverse}_I_{-\text{minus}_M}_{\text{-big}} (M : 'M_m) : \text{(exists} \ n, M^n = 0) \rightarrow (1 - M) \ast m (\sum_{0\leq i<n} M^i) = 1.$
## Starting the proof

<table>
<thead>
<tr>
<th>Goals and Subgoals</th>
<th>Proof-Steps (Tactics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall (M : M_n)(m : \text{nat}), M^m = 0 \implies (1 - M) \times \sum_{i=0}^{m-1} M^i = 1 )</td>
<td>move =&gt; M m nilpotent.</td>
</tr>
<tr>
<td>( (1 - M) \times \sum_{i=0}^{m-1} M^i = 1 )</td>
<td>rewrite big_distr mulmxBr mul1mx.</td>
</tr>
<tr>
<td>( \sum_{i=0}^{m-1} M^i - M^{i+1} )</td>
<td>case : n.</td>
</tr>
<tr>
<td>( \forall (M : M_0)(m : \text{nat}), M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1} )</td>
<td>by rewrite !thinmx0.</td>
</tr>
<tr>
<td>( \forall (M : M_{n+1})(m : \text{nat}), M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1} )</td>
<td></td>
</tr>
</tbody>
</table>
Theorem (Fundamental Lemma of Persistent Homology)

$$\beta_{i}^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{Z}$$

$$\beta_{n}^{k,l} - \beta_{n}^{k,m} = \sum_{1 \leq i \leq k \atop l < j \leq m} (\beta_{n}^{i,p-1} - \beta_{n}^{j,p}) - (\beta_{n}^{i-1,p-1} - \beta_{n}^{j-1,p})$$
Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

\[
\beta_{i,k}^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}
\]

\[
\beta_{n}^{k,l} - \beta_{n}^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{j-1,p-1} - \beta_{n}^{j,p}) - (\beta_{n}^{j-1,p-1} - \beta_{n}^{j-1,p})
\]

Proof

\[
\sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{j-1,i-1} - \beta_{n}^{j,i}) - (\beta_{n}^{j-1,i-1} - \beta_{n}^{j-1,i}) = \]

Theorem (Fundamental Lemma of Persistent Homology)

\[ \beta^j_{i,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{Z} \]

\[ \beta^k_{n,l} - \beta^k_{n,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} \left( \beta^j_{n,p-1} - \beta^j_{n,p} \right) - \left( \beta^{j-1}_{n,p-1} - \beta^{j-1}_{n,p} \right) \]

Proof

\[ \sum_{1 \leq i \leq k} \sum_{l < j \leq m} \left( \beta^j_{n,i-1} - \beta^j_{n,i} \right) - \left( \beta^{j-1}_{n,i-1} - \beta^{j-1}_{n,i} \right) = \]

\[ \sum_{1 \leq i \leq k} \left( \left( \beta^{l+1,i-1} - \beta^{l+1,i} \right) - \left( \beta^{l,i-1} - \beta^{l,i} \right) + \right. \]

\[ \left. \left( \beta^{l+2,i-1} - \beta^{l+2,i} \right) - \left( \beta^{l+1,i-1} - \beta^{l+1,i} \right) + \right. \]

\[ \cdots + \]

\[ \left( \beta^{m-1,i-1} - \beta^{m-1,i} \right) - \left( \beta^{m-2,i-1} - \beta^{m-2,i} \right) + \]

\[ \left( \beta^{m,i-1} - \beta^{m,i} \right) - \left( \beta^{m-1,i-1} - \beta^{m-1,i} \right) \]
Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

\[ \beta_{i,j,k}^{l,m} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{Z} \]

\[ \beta_{n}^{k,l} - \beta_{n}^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{j,p-1} - \beta_{n}^{j,p}) - (\beta_{n}^{j-1,p-1} - \beta_{n}^{j-1,p}) \]

Proof

\[ \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{j,i-1} - \beta_{n}^{j,i}) - (\beta_{n}^{j-1,i-1} - \beta_{n}^{j-1,i}) = \]

\[ \sum_{1 \leq i \leq k} ((\beta_{n}^{l+1,i-1} - \beta_{n}^{l+1,i}) - (\beta_{n}^{l+1,i-1} - \beta_{n}^{l+1,i})+) \]

\[ (\beta_{n}^{l+2,i-1} - \beta_{n}^{l+2,i}) - (\beta_{n}^{l+1,i-1} - \beta_{n}^{l+1,i})+) \]

\[ \cdots \]

\[ (\beta_{n}^{m-1,i-1} - \beta_{n}^{m-1,i}) - (\beta_{n}^{m-2,i-1} - \beta_{n}^{m-2,i})+) \]

\[ (\beta_{n}^{m,i-1} - \beta_{n}^{m,i}) - (\beta_{n}^{m-1,i-1} - \beta_{n}^{m-1,i})+) \]
Theorem (Fundamental Lemma of Persistent Homology)

\[ \beta_{i}^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z} \]

\[ \beta_{n}^{k,l} - \beta_{n}^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{i,p-1} - \beta_{n}^{i,p}) - (\beta_{n}^{i-1,p-1} - \beta_{n}^{i-1,p}) \]

Proof

\[ \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{i-1} - \beta_{n}^{i}) - (\beta_{n}^{i-1,i-1} - \beta_{n}^{i-1,i}) = \]

\[ \sum_{1 \leq i \leq k} ((\beta_{n}^{i-1} - \beta_{n}^{i}) - (\beta_{n}^{i,i} - \beta_{n}^{i})) = \]

\[ \sum_{1 \leq i \leq k} ((\beta_{n}^{i-1} - \beta_{n}^{i}) - (\beta_{n}^{i,i} - \beta_{n}^{i})) + \]

\[ (\beta_{n}^{m-1,i-1} - \beta_{n}^{m-1,i}) - (\beta_{n}^{m-2,i-1} - \beta_{n}^{m-2,i}) + \]

\[ (\beta_{n}^{m,i} - \beta_{n}^{m,i}) - (\beta_{n}^{m-1,i} - \beta_{n}^{m-1,i})) \]
Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

\[ \beta_{i}^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{Z} \]

\[ \beta_{n}^{k,l} - \beta_{n}^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_{n}^{j,i-1} - \beta_{n}^{j+i}) - (\beta_{n}^{j-1,i-1} - \beta_{n}^{j-1,i}) = \ldots \]
Lemma

If $g : \mathbb{N} \to \mathbb{Z}$, then

$$\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)$$
More Examples

Detecting patterns across mathematical libraries

Suggestions provided by ML4PG

Lemma

If \( g : \mathbb{N} \rightarrow \mathbb{Z} \), then

\[
\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)
\]

Proof

\[
\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = \]

Katya and Jónathan (Dundee)  Statistical Machine Learning in ITP  8 November 2013  29 / 40
Suggestions provided by ML4PG

**Lemma**

If $g : \mathbb{N} \rightarrow \mathbb{Z}$, then

$$\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)$$

**Proof**

$$\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(1) - g(0) + g(2) - g(1) + \ldots + g(k + 1) - g(k)$$
Lemma

If $g : \mathbb{N} \rightarrow \mathbb{Z}$, then

$$\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)$$

Proof

$$\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(1) - g(0) + g(2) - g(1) + \ldots + g(k + 1) - g(k)$$
Lemma

If \( g : \mathbb{N} \to \mathbb{Z} \), then

\[
\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)
\]

Proof

\[
\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(1) - g(0) + g(2) - g(1) + \ldots + g(k + 1) - g(k)
\]
Lemma

If \( g : \mathbb{N} \to \mathbb{Z} \), then

\[
\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)
\]

Proof

\[
\sum_{0 \leq i \leq k} (g(i + 1) - g(i)) = g(k + 1) - g(0)
\]
Suggestions provided by ML4PG

Proof Strategy

Apply case on $n$.

1. Prove the base case (a simple task).

2. Prove the case $0 < n$:
   1. expand the summation,
   2. cancel the terms pairwise,
   3. the only terms remaining after the cancellation are the first and the last one.

Lemma

Let $M$ be a nilpotent matrix, then

$$(1 - M) \times \sum_{0 \leq i < n} M^i = 1$$

where $n$ is such that $M^n = 0$
Suggestions provided by ML4PG

Proof Strategy

Apply case on \( n \).

1. Prove the base case (a simple task).
2. Prove the case \( 0 < n \):
   1. expand the summation,
   2. cancel the terms pairwise,
   3. the only terms remaining after the cancellation are the first and the last one.

Lemma

Let \( M \) be a nilpotent matrix, then

\[
(1 - M) \times \sum_{0 \leq i < n} M^i = 1
\]

where \( n \) is such that \( M^n = 0 \)

Proof

\[
(1 - M) \times \sum_{0 \leq i < n} M^i =
\]
**Proof Strategy**

**Apply case on** \( n \).

1. **Prove the base case (a simple task).**
2. **Prove the case** \( 0 < n \):
   1. expand the summation,
   2. cancel the terms pairwise,
   3. the only terms remaining after the cancellation are the first and the last one.

**Lemma**

Let \( M \) be a nilpotent matrix, then

\[
(1 - M) \times \sum_{0 \leq i < n} M^i = 1
\]

where \( n \) is such that \( M^n = 0 \)

**Proof**

\[
(1 - M) \times \sum_{0 \leq i < n} M^i = \\
\sum_{0 \leq i < n} M^i - M^{i+1}
\]
Suggestions provided by ML4PG

Proof Strategy

Apply case on $n$.

1. Prove the base case (a simple task).
2. Prove the case $0 < n$:
   1. expand the summation,
   2. cancel the terms pairwise,
   3. the only terms remaining after the cancellation are the first and the last one.

Lemma

Let $M$ be a nilpotent matrix, then

$$ (1 - M) \times \sum_{0 \leq i < n} M^i = 1 $$

where $n$ is such that $M^n = 0$

Proof

$$ (1 - M) \times \sum_{0 \leq i < n} M^i = \sum_{0 \leq i < n} M^i - M^{i+1} $$

$$ = M^0 - M^1 + M^1 - M^2 + \ldots + M^{n-1} - M^n $$
Suggestions provided by ML4PG

Proof Strategy

Apply case on $n$.

1. Prove the base case (a simple task).
2. Prove the case $0 < n$:
   1. expand the summation,
   2. cancel the terms pairwise,
   3. the only terms remaining after the cancellation are the first and the last one.

Lemma

Let $M$ be a nilpotent matrix, then

$$ (1 - M) \times \sum_{0 \leq i < n} M^i = 1 $$

where $n$ is such that $M^n = 0$

Proof

$$ (1 - M) \times \sum_{0 \leq i < n} M^i = \sum_{0 \leq i < n} M^i - M^{i+1} = M^0 - M^1 + M^2 - M^3 + \ldots + M^{n-1} - M^n $$
Suggestions provided by ML4PG

Proof Strategy

Apply case on $n$.

1. Prove the base case (a simple task).
2. Prove the case $0 < n$:
   - expand the summation,
   - cancel the terms pairwise,
   - the only terms remaining after the cancellation are the first and the last one.

Lemma

Let $M$ be a nilpotent matrix, then

\[
(1 - M) \times \sum_{0 \leq i < n} M^i = 1
\]

where $n$ is such that $M^n = 0$

Proof

\[
(1 - M) \times \sum_{0 \leq i < n} M^i = \sum_{0 \leq i < n} M^i - M^{i+1} = M^0 - M^n = M^0 = 1
\]
An unusual discovery

**Lemma**

Let $M$ be a nilpotent matrix, then there exists $N$ such that $N \times (1 - M) = 1$
An unusual discovery

Lemma

Let $M$ be a nilpotent matrix, then there exists $N$ such that $N \times (1 - M) = 1$

<table>
<thead>
<tr>
<th>Goals and Subgoals</th>
<th>Proof-Steps (Tactics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall (M : M_n)(m : \text{nat}), M^m = 0 \implies \exists N, N \times (1 - M) = 1$</td>
<td>$\text{move} \Rightarrow M \text{ nilpotent.}$</td>
</tr>
<tr>
<td>$\exists N, N \times (1 - M) = 1$</td>
<td>$\text{exists}$</td>
</tr>
<tr>
<td>$(\sum_{i=0}^{m-1} M^i) \times (1 - M)$</td>
<td>$\sum_{0 \leq i &lt; m + 1} (\text{pot_matrix } M \ i)$</td>
</tr>
<tr>
<td>$\sum_{i=0}^{m-1} M^i - M^{i+1}$</td>
<td>$\text{rewrite } \text{big_distr}l \ \text{mulmxrB} \ \text{mulmx1}.$</td>
</tr>
<tr>
<td>$\forall (M : M_0)(m : \text{nat}), M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1}$</td>
<td>$\text{case} : \text{n.}$</td>
</tr>
<tr>
<td>$\forall (M : M_{n+1})(m : \text{nat}), M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1}$</td>
<td>$\text{by rewrite } !\text{thinmx}0.$</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. ML4PG: “Machine Learning for Proof General”
3. Using ML4PG
4. More Examples
   - Detecting patterns across mathematical libraries
   - Detecting irrelevant libraries
5. Conclusions and Further work
An example coming from Game Theory

An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves.
An example coming from Game Theory

An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves. Each internal node is owned by a player and a play of a game is a path from the root to a leaf.
An example coming from Game Theory

An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves. Each internal node is owned by a player and a play of a game is a path from the root to a leaf. A strategy is a game where each internal node has chosen a child.
An example coming from Game Theory

An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves.
Each internal node is owned by a player and a play of a game is a path from the root to a leaf.
A strategy is a game where each internal node has chosen a child.
An example coming from Game Theory

An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves. Each internal node is owned by a player and a play of a game is a path from the root to a leaf.

A strategy is a game where each internal node has chosen a child.

A Nash equilibrium is a strategy in which no agent can change one or more of his choices to obtain a better result.
An example coming from Game Theory

An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves. Each internal node is owned by a player and a play of a game is a path from the root to a leaf. A strategy is a game where each internal node has chosen a child.

A Nash equilibrium is a strategy in which no agent can change one or more of his choices to obtain a better result. A strategy is a subgame perfect equilibrium if it represents Nash equilibrium of every subgame of the original game.
All sequential games have Nash equilibrium.
Formalisations in Coq

All sequential games have Nash equilibrium.

Binary case:

Formalisations in Coq

All sequential games have Nash equilibrium.

**Binary case:**


**General case:**

Formalisations in Coq

All sequential games have Nash equilibrium.
Binary case:


General case:


Is it possible to reuse patterns between these libraries?
All sequential games have Nash equilibrium.

Binary case:


General case:


Is it possible to reuse patterns between these libraries? It is natural to think so, but ...
Formalisations are just too different

Subgame Perfect Equilibrium implies Nash Equilibrium:

<table>
<thead>
<tr>
<th>Binary case</th>
<th>General case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemma SGP_is_NashEq :</td>
<td>Lemma SPE_is_Eq :</td>
</tr>
<tr>
<td>forall s : Strategy, SGP s -&gt; NashEq s.</td>
<td>forall s : Strat, SPE s -&gt; Eq s.</td>
</tr>
<tr>
<td>Proof.</td>
<td>Proof.</td>
</tr>
<tr>
<td>induction s.</td>
<td>intros. destruct s; simpl in H; tauto.</td>
</tr>
<tr>
<td>unfold NashEq. intros _. induction s’.</td>
<td>Qed.</td>
</tr>
<tr>
<td>intros. unfold stratPO. unfold agentConv in H.</td>
<td></td>
</tr>
<tr>
<td>rewrite (H a). trivial.</td>
<td></td>
</tr>
<tr>
<td>unfold agentConv. intros. contradiction.</td>
<td></td>
</tr>
<tr>
<td>unfold SGP. intros [_. _. done]. trivial.</td>
<td>Qed.</td>
</tr>
</tbody>
</table>
Formalisations are just too different

Subgame Perfect Equilibrium implies Nash Equilibrium:

<table>
<thead>
<tr>
<th>Binary case</th>
<th>General case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemma SGP_is_NashEq :</td>
<td>Lemma SPE_is_Eq :</td>
</tr>
<tr>
<td>forall s : Strategy, SGP s -&gt; NashEq s.</td>
<td>forall s : Strat, SPE s -&gt; Eq s.</td>
</tr>
<tr>
<td>Proof.</td>
<td>Proof.</td>
</tr>
<tr>
<td>induction s.</td>
<td>intros. destruct s; simpl in H; tauto.</td>
</tr>
<tr>
<td>unfold NashEq. intro _. induction s’.</td>
<td>Qed.</td>
</tr>
<tr>
<td>intros. unfold stratPO. unfold agentConv in H.</td>
<td></td>
</tr>
<tr>
<td>rewrite (H a). trivial.</td>
<td></td>
</tr>
<tr>
<td>unfold agentConv. intro. contradiction.</td>
<td></td>
</tr>
<tr>
<td>unfold SGP. intro [_. done]. trivial.</td>
<td></td>
</tr>
<tr>
<td>Qed.</td>
<td></td>
</tr>
</tbody>
</table>

No correlation among important theorems of the 2 libraries: completely different datastructures and strategies to prove lemmas.
Formalisations are just too different

Subgame Perfect Equilibrium implies Nash Equilibrium:

<table>
<thead>
<tr>
<th>Binary case</th>
<th>General case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemma SGP_is_NashEq :</td>
<td>Lemma SPE_is_Eq :</td>
</tr>
<tr>
<td>forall s : Strategy, SGP s -&gt; NashEq s.</td>
<td>forall s : Strat, SPE s -&gt; Eq s.</td>
</tr>
<tr>
<td>Proof.</td>
<td>Proof.</td>
</tr>
<tr>
<td>induction s.</td>
<td>intros. destruct s; simpl in H; tauto.</td>
</tr>
<tr>
<td>unfold NashEq. intros _. induction s’.</td>
<td>Qed.</td>
</tr>
<tr>
<td>intros. unfold stratPO. unfold agentConv in H.</td>
<td></td>
</tr>
<tr>
<td>rewrite (H a). trivial.</td>
<td></td>
</tr>
<tr>
<td>unfold agentConv. intros. contradiction.</td>
<td></td>
</tr>
<tr>
<td>unfold SGP. intros [<em>. [</em>. done]]. trivial.</td>
<td></td>
</tr>
<tr>
<td>Qed.</td>
<td></td>
</tr>
</tbody>
</table>

No correlation among important theorems of the 2 libraries: completely different datastructures and strategies to prove lemmas. ML4PG discovers the absence of patterns.
Comparison of the two examples

Orthogonal examples:

- Nilpotent matrices example:
  - Completely unrelated libraries, but common proof strategy.

- Nash example:
  - Similar results, but completely different proof strategies.
Outline

1. Introduction
2. ML4PG: “Machine Learning for Proof General”
3. Using ML4PG
4. More Examples
5. Conclusions and Further work
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
- ML4PG does not assume any knowledge of machine-learning interfaces from the user;
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
- ML4PG does not assume any knowledge of machine-learning interfaces from the user;
- modular: allows the user to make choices regarding approach to levels of proofs and particular statistical algorithms;
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
- ML4PG does not assume any knowledge of machine-learning interfaces from the user;
- modular: allows the user to make choices regarding approach to levels of proofs and particular statistical algorithms;
- tolerant to mixing and matching different proof libraries and different notation used in proofs across different users.
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
- ML4PG does not assume any knowledge of machine-learning interfaces from the user;
- Modular: allows the user to make choices regarding approach to levels of proofs and particular statistical algorithms;
- Tolerant to mixing and matching different proof libraries and different notation used in proofs across different users.
- Easily extendable: e.g. from Coq to SSReflect and ACL2.
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
- ML4PG does not assume any knowledge of machine-learning interfaces from the user;
- modular: allows the user to make choices regarding approach to levels of proofs and particular statistical algorithms;
- tolerant to mixing and matching different proof libraries and different notation used in proofs across different users.
- easily extendable: e.g. from Coq to SSReflect and ACL2.

Most amazingly...
Summary: Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user’s demand;
- ML4PG does not assume any knowledge of machine-learning interfaces from the user;
- modular: allows the user to make choices regarding approach to levels of proofs and particular statistical algorithms;
- tolerant to mixing and matching different proof libraries and different notation used in proofs across different users.
- easily extendable: e.g. from Coq to SSReflect and ACL2.

Most amazingly...

it really works!!!!
Conclusions and Further work

Conclusions

- We can, and perhaps should, apply statistical machine-learning in theorem proving;
- The general task is to use it to process “big data”, or for distributed/collaborative proving.
- Conceptualisation of ML4PG output is a challenge.
Conclusions and Further work

Conclusions

- We can, and perhaps should, apply statistical machine-learning in theorem proving;
- The general task is to use it to process “big data”, or for distributed/collaborative proving.
- Conceptualisation of ML4PG output is a challenge.

Related Work

- ACL2(ml) works as ML4PG in the ACL2 prover and also conceptualise new lemmas. Part of SICSA industrial grant.
Statistical Machine Learning in Interactive Theorem Proving

Katya Komendantskaya and Jonathan Heras
(Funded by EPSRC First Grant Scheme)

University of Dundee

8 November 2013