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

Outline

1. Introduction

2. ML4PG: “Machine Learning for Proof General”

Outline

1. Introduction
2. ML4PG: "Machine Learning for Proof General"
3. Using ML4PG

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.

G. Gonthier. Formal proof - the four-color theorem. Notices of the American Mathematical Society, 55(11):13821393, 2008.

G. Gonthier et al. A Machine-Checked Proof of the Odd Order Theorem. In 4th Conference on Interactive Theorem Proving (ITP13), volume 7998 of Lecture Notes in Computer Science, pages 163179, 2013.

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.

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.

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:

H. Liu and J S. Moore. Executable JVM model for analytic reasoning: a study. Journal Science of Computer Programming - Special issue on advances in interpreters, virtual machines and emulators (IVME'03), 57(3):253–274, 2003.

Computing 5!

Java code:

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

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*

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:

0

stack:

				...

local variables:

5				...

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:

1

stack:

1				...

local variables:

5				...

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:

2

stack:

				...

local variables:

5	1			...

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:

3

stack:

5				...

local variables:

5	1			...

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:

4

stack:

				...

local variables:

5	1			...

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:

5

stack:

1				...

local variables:

5	1			...

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:

6

stack:

5	1			...

local variables:

5	1			...

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				...

local variables:

5	1			...

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:

8

stack:

				...

local variables:

5	5			...

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:

9

stack:

5				...

local variables:

5	5			...

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:

10

stack:

1	5			...

local variables:

5	5			...

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:

11

stack:

4				...

local variables:

5	5			...

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:

12

stack:

				...

local variables:

4	5			...

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:

2

stack:

				...

local variables:

4	5			...

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:

...

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:
13

stack:

0				...

local variables:

0	120			...

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:

14

stack:

120				...

local variables:

0	120			...

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:

15

stack:

120				...

local variables:

0	120			...

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:

15

stack:

120				...

local variables:

0	120			...

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.

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.

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

```
Definition theta_fact (n : nat) := n`!.
```

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)

```
Fixpoint helper_fact (n a : nat) :=
match n with
| 0 => a
| S p => helper_fact p (n * a)
end.

Definition fn_fact (n : nat) :=
   helper_fact n 1.
```

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

```
Lemma fn_fact_is_theta n :
    fn_fact n = theta_fact n.
```

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

```
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

```
Fixpoint loop_sched_fact (n : nat) :=
match n with
| 0 => nseq 3 0
| S p => nseq 11 0 ++ loop_sched_fact p
end.


Definition sched_fact (n : nat) :=
   nseq 2 0 ++ loop_sched_fact n.
```

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

```
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

```
Theorem total_correctness_fact n sf :
  sf = run (sched_fact n)
  (make_state 0 [::n] [::] pi_fact) ->
  next_inst sf = (HALT,0%Z) /\
  top (stack sf) = (n`!).
```

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

```
Lemma fn_fact_is_theta n :
   fn_fact n = theta_fact n.
```

Proof of lemma `fn_fact_is_theta`

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

```
-U:**-  lists.v       All L1     (Coq Script(0) Holes)----------------------------
```

```
1 subgoals, subgoal 1 (ID 13)

============================
 forall n : nat, fn_fact n = theta_fact n
```

```
-U:%%-  *response*     All L1     (Coq Response)------------------------------------
```

Proof of lemma `fn_fact_is_theta`

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

-U:**- lists.v   All L1   (Coq Script(0) Holes)---------------------------

```
1 subgoals, subgoal 1 (ID 14)

n : nat
============================
fn_fact n = theta_fact n
```

-U:%%- *response*   All L1   (Coq Response)-----------------------------------

Proof of lemma fn_fact_is_theta

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

-U:**- lists.v All L1 (Coq Script(0) Holes)----------------------------

```
1 subgoals, subgoal 1 (ID 14)

n : nat
============================
 helper_fact n 1 = n`!
```

-U:%%- *response* All L1 (Coq Response)-----------------------------------

Proof of lemma `fn_fact_is_theta`

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

-U:**- lists.v All L1 (Coq Script(0) Holes)------------------------------

```
1 subgoals, subgoal 1 (ID 14)

n : nat
============================
 helper_fact n 1 = n`!
```

-U:%%- *response* All L1 (Coq Response)-------------------------------------

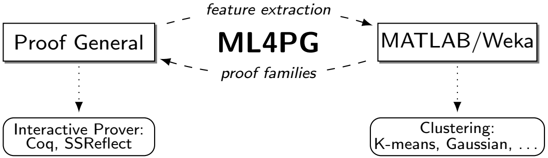
...and now?

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

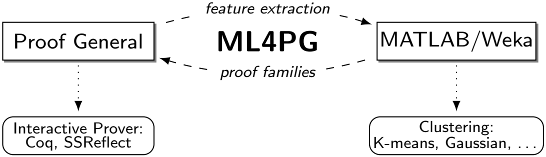
Machine Learning 4 Proof General: interfacing interfaces

...in [2013, Postproc. of UITP'12]



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

1. 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.

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

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

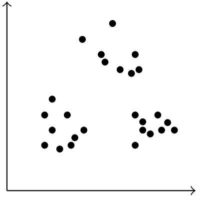
ML4PG approach to proof-clustering

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:

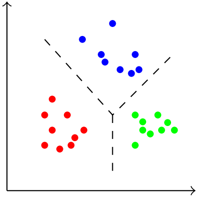
- Unsupervised machine learning technique:



ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

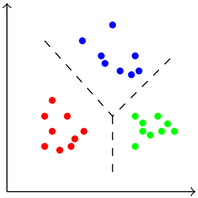


- Engines: Matlab, Weka, Octave, R, . . .

ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

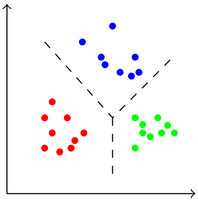


- Engines: Matlab, Weka, Octave, R, . . .

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:

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

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

-U:**- **lists.v** All L1 (Coq Script(0) Holes)----------------------------

```
1 subgoals, subgoal 1 (ID 14)

n : nat
============================
 helper_fact n 1 = n`!
```

-U:%%- ***response*** All L1 (Coq Response)-------------------------------------

Continuation of proof of lemma `fn_fact_is_theta`

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

-U:**- **lists.v** All L1 (Coq Script(0) Holes)---------------------------

```
1 subgoals, subgoal 1 (ID 14)

n : nat
============================
 helper_fact n 1 = n`!
```

-U:%%- ***response*** All L1 (Coq Response)------------------------------------

Continuation of proof of lemma `fn_fact_is_theta`

emacs@joheras-HP-Compaq-6730b-GW687AV

File Edit Options Buffers Tools Coq Proof-General Holes Help

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

-U:**- lists.v All L1 (Coq Script(0) Holes)

```
1 subgoals, subgoal 1 (ID 14)

n : nat
============================
 helper_fact n 1 = n`!
```

```
----------------------------------

Lemma fn_fact_is_theta is similar
to lemmas:
  - fn_expt_is_theta
  - fn_mult_is_theta
  - fn_power_is_theta

----------------------------------
```

-U:%%- *response* All L1 (Coq Response)

Proving lemma fn_fact_is_theta by analogy

Factorial	Exponentiation
`Lemma fn_fact_is_theta n : fn_fact n = n`!.` `Proof.` `move => n. rewrite /fn_fact.`	`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 fn_fact_is_theta by analogy

Factorial	Exponentiation
`Lemma fn_fact_is_theta n : fn_fact n = n`!.` `Proof.` `move => n. rewrite /fn_fact.`	`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_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 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 fn_fact_is_theta by analogy

Factorial	Exponentiation
<pre>Lemma fn_fact_is_theta n : fn_fact n = n`!. Proof. move => n. rewrite /fn_fact. by rewrite helper_fact_is_theta mul1n. Qed.  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.</pre>	<pre>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.</pre>

Proving lemma fn_fact_is_theta by analogy

Factorial	Exponentiation
<pre>Lemma fn_fact_is_theta n : fn_fact n = n`!. Proof. move => n. rewrite /fn_fact. by rewrite helper_fact_is_theta mul1n. Qed.  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.</pre>	<pre>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.</pre>

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

Algorithm:	$g = 1$ $(n = 16)$	$g = 2$ $(n = 18)$	$g = 3$ $(n = 21)$	$g = 4$ $(n = 24)$	$g = 5$ $(n = 29)$
K-means	$30^{a,b,d}$	$\mathbf{4}^{a-d}$	$\mathbf{4}^{a-d}$	$2^{c,d}$	0
E.M.	21^{a-d}	7^{a-d}	7^{a-d}	0	0
FarthestFirst	28^{a-d}	25^{a-d}	0	0	0

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 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.
```

Proving lemma program_is_fn_fact by analogy

Exponentiation (ML4PG suggestion)

```
Lemma program_is_fn_expt n m :
run (sched_expt n m)(make_state 0 [::n;m] [::] pi_expt)=
(make_state 14 [::0;fn_expt n m] (push (fn_expt n m)[::])pi_expt).
Proof.
rewrite run_app loop_is_helper_expt.
Qed.

Lemma loop_is_helper_expt n m a :
run (loop_sched_expt n)(make_state 2 [::n;m;a] [::] pi_expt)=
(make_state 14 [::0;(helper_expt n m a)] (push (helper_expt n m a)[::])pi_expt)
Proof.
move : n a; elim : m => [// | m IH n a].
by rewrite -IH subn1 -pred_Sn.
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.


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 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.
```

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.

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.
```

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 = run (sched_fact n)(make_state 0 [::n] [::] pi_fact)->
next_inst sf = (HALT,0\%Z)/\ top (stack sf)= (n`!).
Proof.
move => H; split
```

Proving total correctness by analogy

Exponentiation (ML4PG suggestion)

```
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 = run (sched_fact n)(make_state 0 [::n] [::] pi_fact)->
next_inst sf = (HALT,0\%Z)/\ top (stack sf)= (n`!).
Proof.
move => H; split
    ; rewrite H program_is_fn_fact fn_fact_is_theta.
Qed.
```

Proving total correctness by analogy

Factorial

```
Theorem total_correctness_fact n sf :
sf = run (sched_fact n)(make_state 0 [::n] [::] pi_fact)->
next_inst sf = (HALT,0\%Z)/\ top (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.

Proving total correctness by analogy

Factorial

```
Theorem total_correctness_fact n sf :
sf = run (sched_fact n)(make_state 0 [::n] [::] pi_fact)->
next_inst sf = (HALT,0\%Z)/\ top (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:

$$\sum_{0 \leq i < 2n | odd\ i} i = n^2, \prod_{0 \leq i \leq n} i = n!, \bigcup_{i \in I} f(i), \ldots$$

The bigop library

- SSREFLECT library about indexed big “operations”
- Examples:

$$\sum_{0 \le i < 2n | odd\ i} i = n^2, \prod_{0 \le i \le n} i = n!, \bigcup_{i \in I} f(i), \ldots$$

- 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 $M^n = 0$

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 (M : 'M_m) : (exists n, M^n = 0) ->
  (1 - M) *m (\sum_(0<=i<n) M^i) = 1.
```

Starting the proof

Goals and Subgoals	Proof-Steps (Tactics)
$\forall\ (M : M_n)(m : nat),\ M^m = 0 \implies (1 - M) \times \sum_{i=0}^{m-1} M^i = 1$	
	`move => M m nilpotent.`
$(1 - M) \times \sum_{i=0}^{m-1} M^i = 1$	
	`rewrite big_distrr mulmxBr mul1mx.`
$\sum_{i=0}^{m-1} M^i - M^{i+1}$	
	`case : n.`
$\forall\ (M : M_0)(m : nat),\ M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1}$	
	`by rewrite !thinmx0.`
$\forall\ (M : M_{n+1})(m : nat),\ M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1}$	

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

$\beta_i^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

$$\beta_n^{k,l} - \beta_n^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p})$$

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

$\beta_i^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

$$\beta_n^{k,l} - \beta_n^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p})$$

Proof

$$\sum_{1 \leq i \leq k} \sum\nolimits_{l < j \leq m} (\beta_n^{j,i-1} - \beta_n^{j,i}) - (\beta_n^{j-1,i-1} - \beta_n^{j-1,i}) \quad =$$

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

$\beta_i^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

$$\beta_n^{k,l} - \beta_n^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p})$$

Proof

$$\begin{aligned}
\sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,i-1} - \beta_n^{j,i}) - (\beta_n^{j-1,i-1} - \beta_n^{j-1,i}) \quad &= \\
\sum_{1 \leq i \leq k} ((\beta_n^{l+1,i-1} - \beta_n^{l+1,i}) - (\beta_n^{l,i-1} - \beta_n^{l,i}) + \\
(\beta_n^{l+2,i-1} - \beta_n^{l+2,i}) - (\beta_n^{l+1,i-1} - \beta_n^{l+1,i}) + \\
\ldots \\
(\beta_n^{m-1,i-1} - \beta_n^{m-1,i}) - (\beta_n^{m-2,i-1} - \beta_n^{m-2,i}) + \\
(\beta_n^{m,i-1} - \beta_n^{m,i}) - (\beta_n^{m-1,i-1} - \beta_n^{m-1,i}))
\end{aligned}$$

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

$\beta_i^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

$$\beta_n^{k,l} - \beta_n^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p})$$

Proof

$$\begin{aligned}
\sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,i-1} - \beta_n^{j,i}) - (\beta_n^{j-1,i-1} - \beta_n^{j-1,i}) \quad &= \\
\sum_{1 \leq i \leq k} ((\beta_n^{l+1,i-1} - \beta_n^{l+1,i}) - (\beta_n^{l,i-1} - \beta_n^{l,i}) + \\
(\beta_n^{l+2,i-1} - \beta_n^{l+2,i}) - (\beta_n^{l+1,i-1} - \beta_n^{l+1,i}) + \\
\ldots \\
(\beta_n^{m-1,i-1} - \beta_n^{m-1,i}) - (\beta_n^{m-2,i-1} - \beta_n^{m-2,i}) + \\
(\beta_n^{m,i-1} - \beta_n^{m,i}) - (\beta_n^{m-1,i-1} - \beta_n^{m-1,i}))
\end{aligned}$$

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

$\beta_i^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

$$\beta_n^{k,l} - \beta_n^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p})$$

Proof

$$\begin{array}{rl} \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,i-1} - \beta_n^{j,i}) - (\beta_n^{j-1,i-1} - \beta_n^{j-1,i}) & = \\ \sum_{1 \leq i \leq k} ((\beta_n^{l+1,i-1} - \beta_n^{l+1,i}) - (\beta_n^{l,i-1} - \beta_n^{l,i}) + & \\ (\beta_n^{l+2,i-1} - \beta_n^{l+2,i}) - (\beta_n^{l+1,i-1} - \beta_n^{l+1,i}) + & \\ \ldots & \\ (\beta_n^{m-1,i-1} - \beta_n^{m-1,i}) - (\beta_n^{m-2,i-1} - \beta_n^{m-2,i}) + & \\ (\beta_n^{m,i-1} - \beta_n^{m,i}) - (\beta_n^{m-1,i-1} - \beta_n^{m-1,i})) & \end{array}$$

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

$\beta_i^{j,k} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$

$$\beta_n^{k,l} - \beta_n^{k,m} = \sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,p-1} - \beta_n^{j,p}) - (\beta_n^{j-1,p-1} - \beta_n^{j-1,p})$$

Proof

$$\sum_{1 \leq i \leq k} \sum_{l < j \leq m} (\beta_n^{j,i-1} - \beta_n^{j,i}) - (\beta_n^{j-1,i-1} - \beta_n^{j-1,i}) = \sum_{1 \leq i \leq k} (\beta_n^{m,i-1} - \beta_n^{m,i}) - (\beta_n^{l,i-1} - \beta_n^{l,i}) = \ldots$$

Suggestions provided by ML4PG

Lemma

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

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

Suggestions provided by ML4PG

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

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

Suggestions provided by ML4PG

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

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

Suggestions provided by ML4PG

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

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

Suggestions provided by ML4PG

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

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

Suggestions provided by ML4PG

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

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

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

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

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

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

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

$$(1 - M) \times \sum_{0 \leq i < n} M^i =$$
$$\sum_{0 \leq i < n} M^i - M^{i+1} =$$
$$M^0 - \cancel{M^1} + \cancel{M^1} - \cancel{M^2} + \ldots + \cancel{M^{n-1}} - M^n$$

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

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

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 $N \times (1 - M) = 1$

Goals and Subgoals	Proof-Steps (Tactics)
$\forall\ (M : M_n)(m : nat), M^m = 0 \implies \exists N, N \times (1 - M) = 1$	`move => M m nilpotent.`
$\exists N, N \times (1 - M) = 1$	`exists \sum_(0<=i<m.+1)(pot_matrix M i).`
$(\sum_{i=0}^{m-1} M^i) \times (1 - M)$	`rewrite big_distrl mulmxrB mulmx1.`
$\sum_{i=0}^{m-1} M^i - M^{i+1}$	`case : n.`
$\forall\ (M : M_0)(m : nat), M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1}$	`by rewrite !thinmx0.`
$\forall\ (M : M_{n+1})(m : nat), M^m = 0 \implies \sum_{i=0}^{m-1} M^i - M^{i+1}$	

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

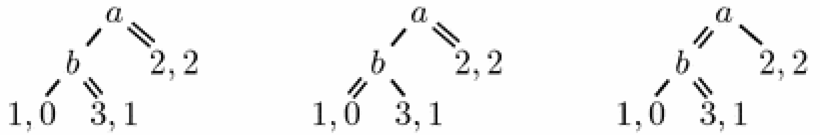
An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves.
Each internal node is owned by a player and a play of a game is a path from the root to a leaf.

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.

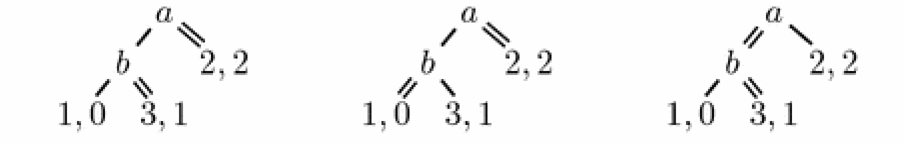
An example coming from Game Theory

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

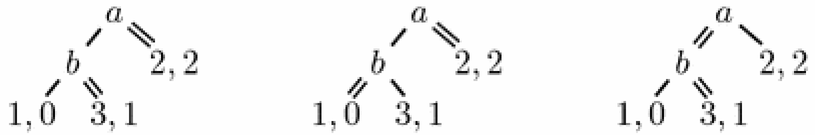
An (abstract) sequential game can be represented as a tree with pay-off functions in the leaves.
Each internal node is owned by a player and a play of a game is a path from the root to a leaf.
A *strategy* is a game where each internal node has chosen a child.



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.

Formalisations in Coq

All sequential games have Nash equilibrium.

Formalisations in Coq

All sequential games have Nash equilibrium.

Binary case:

R. Vestergaard. A constructive approach to sequential nash equilibria. Information Processing Letter, 97:4651, 2006.

Formalisations in Coq

All sequential games have Nash equilibrium.

Binary case:

R. Vestergaard. A constructive approach to sequential nash equilibria. Information Processing Letter, 97:4651, 2006.

General case:

S. Le Roux. Acyclic Preferences and Existence of Sequential Nash Equilibria: A Formal and Constructive Equivalence. In 20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs07), volume 5674 of Lecture Notes in Computer Science, pages 293309, 2009.

Formalisations in Coq

All sequential games have Nash equilibrium.

Binary case:

R. Vestergaard. A constructive approach to sequential nash equilibria. Information Processing Letter, 97:4651, 2006.

General case:

S. Le Roux. Acyclic Preferences and Existence of Sequential Nash Equilibria: A Formal and Constructive Equivalence. In 20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs07), volume 5674 of Lecture Notes in Computer Science, pages 293309, 2009.

Is it possible to reuse patterns between these libraries?

Formalisations in Coq

All sequential games have Nash equilibrium.

Binary case:

R. Vestergaard. A constructive approach to sequential nash equilibria. Information Processing Letter, 97:4651, 2006.

General case:

S. Le Roux. Acyclic Preferences and Existence of Sequential Nash Equilibria: A Formal and Constructive Equivalence. In 20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs07), volume 5674 of Lecture Notes in Computer Science, pages 293309, 2009.

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:

Binary case	General case

`Lemma SGP_is_NashEq :`
`forall s : Strategy, SGP s -> NashEq s.`
`Proof.`
`induction s.`
`unfold NashEq. intros _. induction s'.`
`intros. unfold stratPO. unfold agentConv in H.`
`rewrite (H a). trivial.`
`unfold agentConv. intros. contradiction.`
`unfold SGP. intros [_ [_ done]]. trivial.`
`Qed.`
	`Lemma SPE_is_Eq :`
	`forall s : Strat, SPE s -> Eq s.`
	`Proof.`
	`intros. destruct s; simpl in H; tauto.`
	`Qed.`

Formalisations are just too different

Subgame Perfect Equilibrium implies Nash Equilibrium:

Binary case	General case

`Lemma SGP_is_NashEq :`
`forall s : Strategy, SGP s -> NashEq s.`
`Proof.`
`induction s.`
`unfold NashEq. intros _. induction s'.`
`intros. unfold stratPO. unfold agentConv in H.`
`rewrite (H a). trivial.`
`unfold agentConv. intros. contradiction.`
`unfold SGP. intros [_ [_ done]]. trivial.`
`Qed.`
	`Lemma SPE_is_Eq :`
	`forall s : Strat, SPE s -> Eq s.`
	`Proof.`
	`intros. destruct s; simpl in H; tauto.`
	`Qed.`

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:

Binary case	General case

`Lemma SGP_is_NashEq :`
`forall s : Strategy, SGP s -> NashEq s.`
`Proof.`
`induction s.`
`unfold NashEq. intros _. induction s'.`
`intros. unfold stratPO. unfold agentConv in H.`
`rewrite (H a). trivial.`
`unfold agentConv. intros. contradiction.`
`unfold SGP. intros [_ [_ done]]. trivial.`
`Qed.`
	`Lemma SPE_is_Eq :`
	`forall s : Strat, SPE s -> Eq s.`
	`Proof.`
	`intros. destruct s; simpl in H; tauto.`
	`Qed.`

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