Can statistical machine learning advance mechanised proof technology?

Katya Komendantskaya, joint work with Jonathan Heras
(Funded by EPSRC First Grant Scheme)

University of Dundee

19 June 2013

Outline

1. Motivation: machine-learning for automated theorem proving?

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
   - Proof pattern recognition in ATPs
   - Proof pattern recognition in ITPs

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
   - Proof pattern recognition in ATPs
   - Proof pattern recognition in ITPs
3. More Examples
   - The bigop library
   - The CoqEAL library

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
   - Proof pattern recognition in ATPs
   - Proof pattern recognition in ITPs
3. More Examples
   - The bigop library
   - The CoqEAL library
4. Conclusions and Further work

About myself

- 1998 – 2003 Undergraduate degree in Logic, Moscow State University; (1st class honours, gold medal for excellency).

About myself

- 1998 – 2003 Undergraduate degree in Logic, Moscow State University; (1st class honours, gold medal for excellency).
- 2004 – 2007 PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)

About myself

- 1998 – 2003 Undergraduate degree in Logic, Moscow State University; (1st class honours, gold medal for excellency).
- 2004 – 2007 PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
- 2007 – 2008 First postdoc project with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.

About myself

- 1998 – 2003 Undergraduate degree in Logic, Moscow State University; (1st class honours, gold medal for excellency).
- 2004 – 2007 PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
- 2007 – 2008 First postdoc project with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
- 2008 – 2011 EPSRC TCS research fellowship first in St Andrews, later transferred to the School of Computing in Dundee.

About myself

- 1998 – 2003 Undergraduate degree in Logic, Moscow State University; (1st class honours, gold medal for excellency).
- 2004 – 2007 PhD in the UCC, Ireland. (The University (and department!) of the famous George Boole)
- 2007 – 2008 First postdoc project with Yves Bertot in INRIA, France - on guardedness of corecursive functions in Coq.
- 2008 – 2011 EPSRC TCS research fellowship first in St Andrews, later transferred to the School of Computing in Dundee.
- 2010 – now – Senior Lecturer, School of Computing, University of Dundee. We are growing a new "LiCS" group in Dundee...

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...)
2. Corecursion in Higher-order Interactive Theorem Provers
3. Merging Symbolic and Statistical (Machine Learning) methods
4. Categorical Semantics of Computations

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...) (PhD thesis)
2. Corecursion in Higher-order Interactive Theorem Provers
3. Merging Symbolic and Statistical (Machine Learning) methods
4. Categorical Semantics of Computations

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...) (PhD thesis)
2. Corecursion in Higher-order Interactive Theorem Provers (Postdoc in INRIA)
3. Merging Symbolic and Statistical (Machine Learning) methods
4. Categorical Semantics of Computations

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...) (PhD thesis)
2. Corecursion in Higher-order Interactive Theorem Provers (Postdoc in INRIA)
3. Merging Symbolic and Statistical (Machine Learning) methods (PhD Thesis, first EPSRC fellowship, EPSRC First Grant)
4. Categorical Semantics of Computations

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...) (PhD thesis)
2. Corecursion in Higher-order Interactive Theorem Provers (Postdoc in INRIA)
3. Merging Symbolic and Statistical (Machine Learning) methods (PhD Thesis, first EPSRC fellowship, EPSRC First Grant)
4. Categorical Semantics of Computations (in parallel to the above, joint with Edinburgh, Bath)

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...) (PhD thesis)
2. Corecursion in Higher-order Interactive Theorem Provers (Postdoc in INRIA)
3. Merging Symbolic and Statistical (Machine Learning) methods (PhD Thesis, first EPSRC fellowship, EPSRC First Grant)
4. Categorical Semantics of Computations (in parallel to the above, joint with Edinburgh, Bath)
5. ... recently "Coalgebraic Logic Programming for Type Inference" EPSRC grant (merging 1, 2, 4)

Research interests

My research interests can be classified into four main themes:

1. Logic Programming and its applications (in AI, Automated reasoning, Type Inference ...) (PhD thesis)
2. Corecursion in Higher-order Interactive Theorem Provers (Postdoc in INRIA)
3. Merging Symbolic and Statistical (Machine Learning) methods (PhD Thesis, first EPSRC fellowship, EPSRC First Grant)
4. Categorical Semantics of Computations (in parallel to the above, joint with Edinburgh, Bath)
5. ... recently "Coalgebraic Logic Programming for Type Inference" EPSRC grant (merging 1, 2, 4)

The work presented today is the result of the EPSRC First Grant. (2012-2013)

Why Machine-Learning?

- ... Digital era means most of information (in science, industries, even art!) is stored/handled in electronic form.

Why Machine-Learning?

- ... Digital era means most of information (in science, industries, even art!) is stored/handled in electronic form.
- ... Computer-generated data may not make much sense to human users; or in fact, other computers!

Why Machine-Learning?

- ... Digital era means most of information (in science, industries, even art!) is stored/handled in electronic form.
- ... Computer-generated data may not make much sense to human users; or in fact, other computers!
- The volumes of data make it unfeasible to be processed and interpreted manually.

Why Machine-Learning?

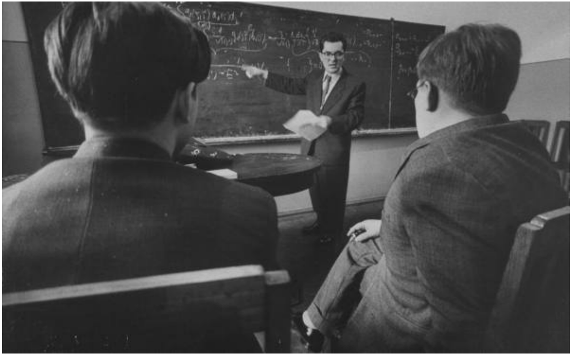
- ... Digital era means most of information (in science, industries, even art!) is stored/handled in electronic form.
- ... Computer-generated data may not make much sense to human users; or in fact, other computers!
- The volumes of data make it unfeasible to be processed and interpreted manually.
- ... the only hope is, our machine-learning algorithms become fast and clever enough to do that dirty (pre-processing) work for us!

Why Machine-Learning?

- ... Digital era means most of information (in science, industries, even art!) is stored/handled in electronic form.
- ... Computer-generated data may not make much sense to human users; or in fact, other computers!
- The volumes of data make it unfeasible to be processed and interpreted manually.
- ... the only hope is, our machine-learning algorithms become fast and clever enough to do that dirty (pre-processing) work for us!

So, why should we (logicians) care?

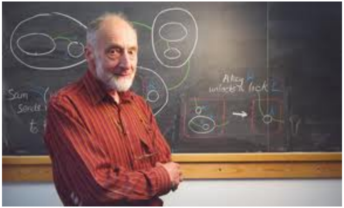
So, why should we (logicians) care?



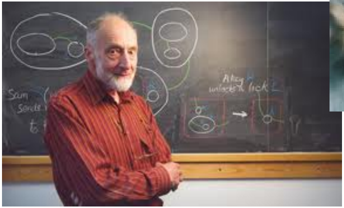
The answer is...

The answer is...

The answer is...



The answer is...



No matter what your personal choice is, . . .

- ... increasingly, theorems [be it mathematics or software/hardware verification] are proven IN automated provers.

No matter what your personal choice is, . . .

- ... increasingly, theorems [be it mathematics or software/hardware verification] are proven IN automated provers.
- Manual handling of various proofs, strategies, libraries, becomes difficult.

No matter what your personal choice is, . . .

- ... increasingly, theorems [be it mathematics or software/hardware verification] are proven IN automated provers.
- Manual handling of various proofs, strategies, libraries, becomes difficult.
- ... team-development is hard, especially that TPs are sensitive to notation;

No matter what your personal choice is, . . .

- ... increasingly, theorems [be it mathematics or software/hardware verification] are proven IN automated provers.
- Manual handling of various proofs, strategies, libraries, becomes difficult.
- ... team-development is hard, especially that TPs are sensitive to notation;
- ... comparison of proofs and proof similarities across libraries or even within one big library are hard;

Main applications in Automated Theorem Proving:

Where can we use ML?

ML in other areas of (Computer) Science:

Where data is abundant, and needs quick automated classification:

- robotics (from space rovers to small apps in domestic appliances, cars...);
- image processing;
- natural language processing;
- web search;
- computer network analysis;
- Medical diagnostics;
- etc, etc, ...

In all these areas, ML is a common tool-of-the-trade, additional to the primary research specialisation.

Will this practice come to Automated theorem proving?

Automated reasoning does NOT need ML applications:

...where AR does not need help

- verification (unlike in Medical diagnosis)
- language parsing (unlike in NLP)

Automated reasoning does NOT need ML applications:

...where AR does not need help

- verification (unlike in Medical diagnosis)
- language parsing (unlike in NLP)

... where we do not trust them

- new theoretical break-troughs (formulation of new theorems);
- giving semantics to data (cf. Deep learning).

So,...

where do we both need ML-tools and trust them?

So,...

where do we both need ML-tools and trust them?

- finding common proof-patterns in proofs across various scripts, libraries, users, notations;

So,...

where do we both need ML-tools and trust them?

- finding common proof-patterns in proofs across various scripts, libraries, users, notations;
- providing proof-hints, especially in (industrial) cases where routine similar cases are frequent, and proof development is distributed across several programmers.

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
   - Proof pattern recognition in ATPs
   - Proof pattern recognition in ITPs
3. More Examples
   - The bigop library
   - The CoqEAL library
4. Conclusions and Further work

ATPs and ITPs

- Automated Theorem Provers (ATPs) and SAT/SMT solvers are
  - ...fast and efficient;
  - ...applied in different contexts: program verification, scheduling, test case generation, etc.

ATPs and ITPs

- Automated Theorem Provers (ATPs) and SAT/SMT solvers are
  - ...fast and efficient;
  - ...applied in different contexts: program verification, scheduling, test case generation, etc.
- Interactive Theorem Provers (ITPs) have been
  - ...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.

Challenges

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

Machine-Learning in the software verification cycle

Consider a sample software verification cycle

Where would we welcome statistical machine-learning assistance?

Running example

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

Running example

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.

Running example

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.

Running example

Java code:

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

Running example

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*

Running example

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				...

Running example

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				...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

Java code:

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

Bytecode:

...

JVM model:

...

Running example

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			...

Running example

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			...

Running example

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			...

Running example

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`!).
```

Introducing machine-learning...

Where and how it is best to apply machine-learning in such cycles is a research question on its own.

Introducing machine-learning...

Where and how it is best to apply machine-learning in such cycles is a research question on its own.

... Your own ideas/suggestions are welcome

Introducing machine-learning...

Where and how it is best to apply machine-learning in such cycles is a research question on its own.

... Your own ideas/suggestions are welcome
It what follows, I'll explain a few existing approaches.

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
   - Proof pattern recognition in ATPs
   - Proof pattern recognition in ITPs
3. More Examples
   - The bigop library
   - The CoqEAL library
4. Conclusions and Further work

Proof pattern recognition in ATPs

Given a proof goal, ATPs apply various lemmas to rewrite or simplify the goal until it is proven.

Proof pattern recognition in ATPs

Given a proof goal, ATPs apply various lemmas to rewrite or simplify the goal until it is proven.

Goal

Apply machine-learning techniques to improve the premise selection procedure on the basis of previous experience.

Proof pattern recognition in ATPs

Given a proof goal, ATPs apply various lemmas to rewrite or simplify the goal until it is proven.

Goal

Apply machine-learning techniques to improve the premise selection procedure on the basis of previous experience.

References:

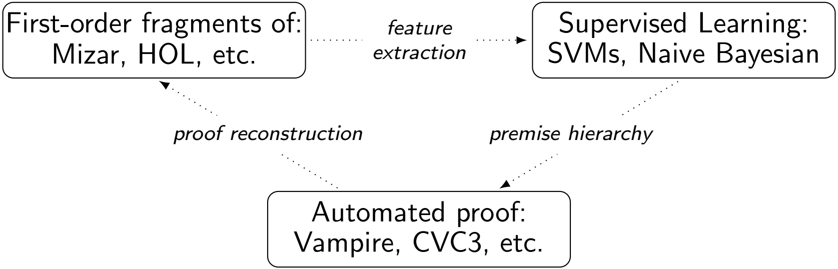
- D. Kühlwein et al. MaSh: Machine Learning for Sledgehammer. In ITP'13, 2013
- C. Kaliszyk and J. Urban. Learning-assisted Automated Reasoning with Flyspeck. 2012
- D. Kühlwein et al. Overview and evaluation of premise selection techniques for large theory mathematics. In IJCAR12, LNCS 7364, pages 378–392, 2012.
- E. Tsivtsivadze et al. Semantic graph kernels for automated reasoning. In SDM11, pages 795–803, 2011.

Application to ITPs

Several ITPs use ATPs to discharge proof obligations. Then, the ATP approach can be used to speed up those proofs.

Application to ITPs

Several ITPs use ATPs to discharge proof obligations. Then, the ATP approach can be used to speed up those proofs.



Intuitive idea

Example Goal

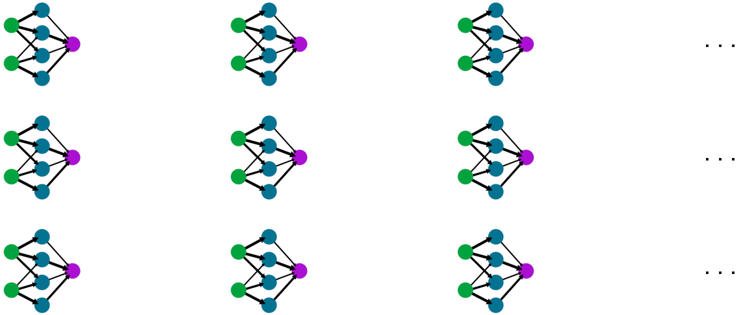
Determine the lemmas that can be useful to prove the equivalence between the recursive and tail-recursive versions of factorial.

Intuitive idea

Example Goal

Determine the lemmas that can be useful to prove the equivalence between the recursive and tail-recursive versions of factorial.

A classifier for each lemma in the library.



Intuitive idea

Example Goal

Determine the lemmas that can be useful to prove the equivalence between the recursive and tail-recursive versions of factorial.

Training phase:

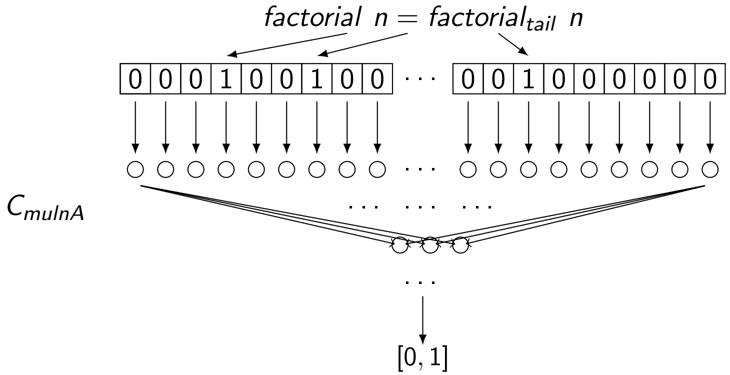
- lemma A is used in the proof of lemma $B \implies < A > (B) = 1$;
- otherwise $\implies < A > (B) = 0$;

Intuitive idea

Example Goal

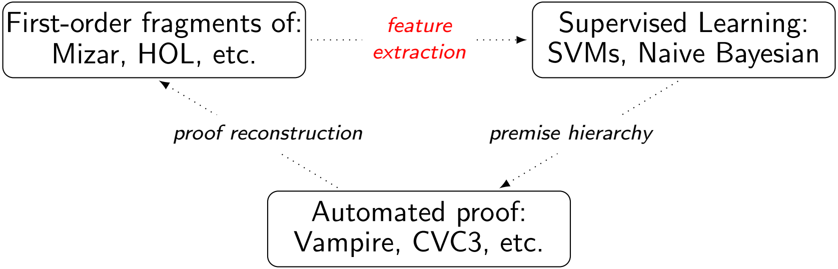
Determine the lemmas that can be useful to prove the equivalence between the recursive and tail-recursive versions of factorial.

Testing phase:



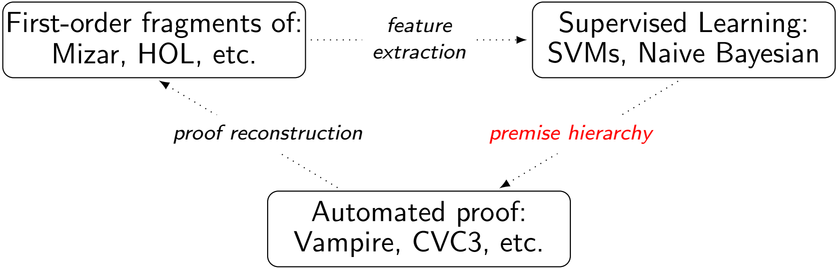
Features of this approach

1. Feature extraction:
   - features are extracted from all first-order formulas of the library;
   - sparse feature vectors (10^6 features);
   - classifier for every lemma of the library.



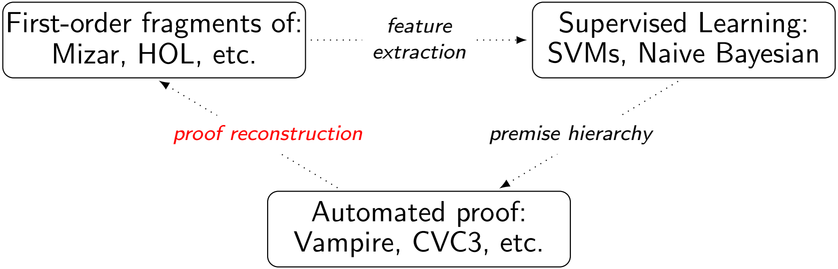
Features of this approach

2. Machine-learning tools:
   - work with supervised learning;
   - algorithms range from SVMs to Naive Bayes learning;
   - sparse methods; using software such as SNoW.



Features of this approach

3. Main improvement:
    - the number of goals proven automatically increases by up to 20% – 40%



Outline

1. Motivation: machine-learning for automated theorem proving?

2. Two main trends: ATP and ITP
   - Proof pattern recognition in ATPs
   - Proof pattern recognition in ITPs

3. More Examples
   - The bigop library
   - The CoqEAL library

4. Conclusions and Further work

Proof pattern recognition in ITPs

In ITPs, the proof development is interactive (via tactics).

Proof pattern recognition in ITPs

In ITPs, the proof development is interactive (via tactics).

Goal: make machine-learning a part of interactive proof development

Apply machine-learning methods to:

- find common proof-patterns in proofs across various scripts, libraries, users and notations;
- and provide proof-hints;
- assist the user, not the prover.

Proof pattern recognition in ITPs

In ITPs, the proof development is interactive (via tactics).

Goal: make machine-learning a part of interactive proof development

Apply machine-learning methods to:

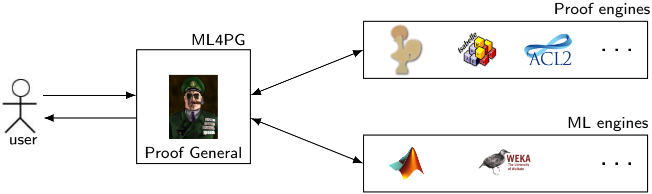
- find common proof-patterns in proofs across various scripts, libraries, users and notations;
- and provide proof-hints;
- assist the user, not the prover.

ML4PG:

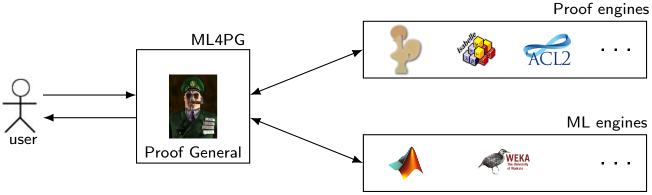
- Proof General extension which applies machine learning methods to Coq/SSReflect proofs. [Now available in standard Proof General distribution!!!]

E. Komendantskaya, J. Heras and G. Grov. Machine learning in Proof General: interfacing interfaces. EPTCS Post-proceedings of User Interfaces for Theorem Provers. 2013.

Overall architecture of ML4PG

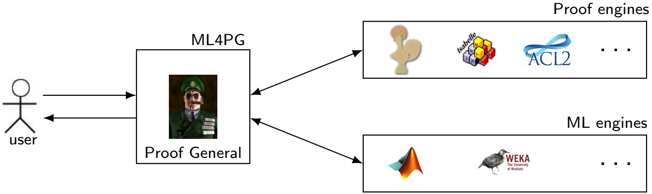


Overall architecture of ML4PG



Interaction with ML4PG:

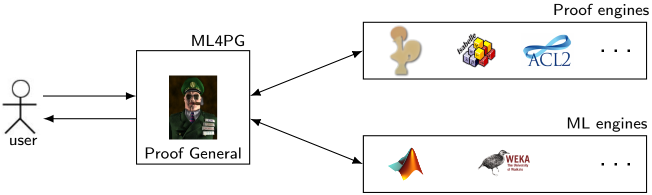
Overall architecture of ML4PG



Interaction with ML4PG:

- User interacts with Proof General as usual,

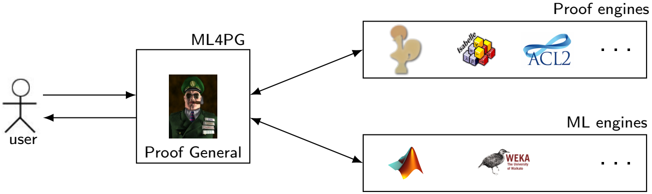
Overall architecture of ML4PG



Interaction with ML4PG:

- User interacts with Proof General as usual,
- User gets stuck in a proof,

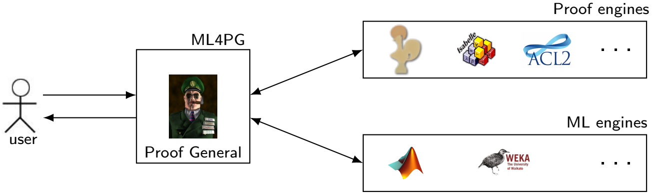
Overall architecture of ML4PG



Interaction with ML4PG:

- User interacts with Proof General as usual,
- User gets stuck in a proof,
- User configures ML4PG,

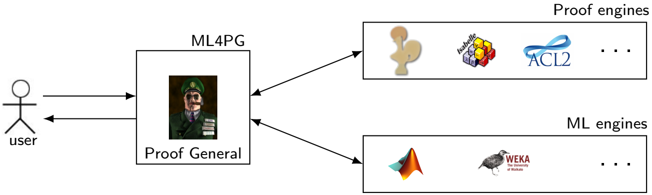
Overall architecture of ML4PG



Interaction with ML4PG:

- User interacts with Proof General as usual,
- User gets stuck in a proof,
- User configures ML4PG,
- User calls for a statistical hint,

Overall architecture of ML4PG



Interaction with ML4PG:

- User interacts with Proof General as usual,
- User gets stuck in a proof,
- User configures ML4PG,
- User calls for a statistical hint,
- ML4PG informs the user of arising proof patterns.

A proof in Coq/SSReflect

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

File Edit Options Buffers Tools Coq Proof-General Holes Help

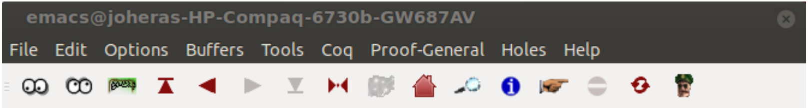
```
Lemma fact_tail_aux_lemma :
forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
```

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

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

  ============================
   forall n a : nat, fact_tail_aux n a = a * n`!
```

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

A proof in Coq/SSReflect




```
Lemma fact_tail_aux_lemma :
forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
move => n.
```

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

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

n : nat
============================
 forall a : nat, fact_tail_aux n a = a * n`!
```

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

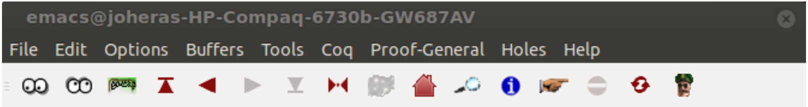
A proof in Coq/SSReflect

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

File Edit Options Buffers Tools Coq Proof-General Holes Help

```
Lemma fact_tail_aux_lemma :
 forall (a n : nat), fact_tail_aux n a = a * n`!.
 Proof.
 move => n. elim : n => [a| n IH a /=].

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

  2 subgoals, subgoal 1 (ID 24)

  a : nat
  ============================
   fact_tail_aux 0 a = a * 0`!

subgoal 2 (ID 28) is:
 fact_tail_aux n (n.+1 * a) = a * (n.+1)`!

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

A proof in Coq/SSReflect




```
Lemma fact_tail_aux_lemma :
 forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
move => n. elim : n => [a| n IH a /=].
  by rewrite /theta_fact fact0 muln1.
```

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

```
1 subgoals, subgoal 1 (ID 28)

n : nat
IH : forall a : nat, fact_tail_aux n a = a * n`!
a : nat
============================
 fact_tail_aux n (n.+1 * a) = a * (n.+1)`!
```

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

Feature extraction mechanism

```
Lemma fact_tail_aux_lemma :
 forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
```

	tactics	N tactics	arg type	tactic arg is hypothesis?	top symbol	subgoals
g1						
g2						
g3						
g4						
g5						

Feature extraction mechanism

```
Lemma fact_tail_aux_lemma :
 forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
move => n.
```

	tactics	*N tactics*	*arg type*	*tactic arg is hypothesis?*	*top symbol*	*subgoals*
g1	move	1	nat	no	forall	1
g2						
g3						
g4						
g5						

Feature extraction mechanism

```
Lemma fact_tail_aux_lemma :
  forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
move => n. elim : n => [a| n IH a /=].
```

	tactics	*N tactics*	*arg type*	*tactic arg is hypothesis?*	*top symbol*	*subgoals*
g1	move	1	nat	no	forall	1
g2	elim, move	2	nat, [nat | nat Prop nat]	yes	forall	2
g3						
g4						
g5						

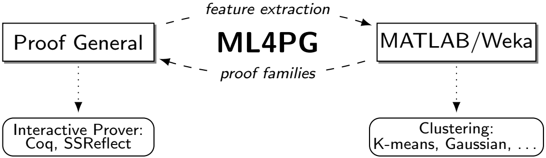
Feature extraction mechanism

```
Lemma fact_tail_aux_lemma :
 forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
move => n. elim : n => [a| n IH a /=].
  by rewrite /theta_fact fact0 muln1.
```

	tactics	*N tactics*	*arg type*	*tactic arg is hypothesis?*	*top symbol*	*subgoals*
g1	move	1	nat	no	forall	1
g2	elim, move	2	nat, [nat | nat Prop nat]	yes	forall	2
g3	rewrite	1	Prop, Prop, Prop	EL_1, EL_2, EL_3	equal	1
g4						
g5						

ML4PG

ML4PG assists the user providing similar lemmas as proof hints.



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;
   - the feature vectors are fixed at the size of 30;
   - longer proofs are analysed by means of the proof-patch method.

Features of this approach

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

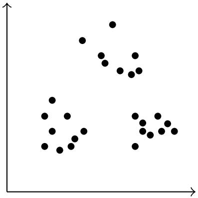
ML4PG approach to proof-clustering

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:

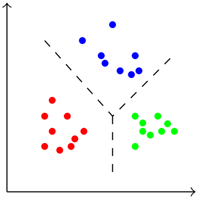
- Unsupervised machine learning technique:



ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

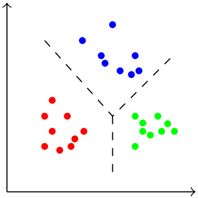


- Engines: Matlab, Weka, Octave, R, ...

ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

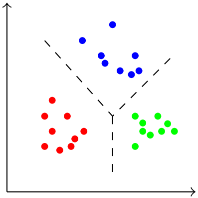


- Engines: Matlab, Weka, Octave, R, . . .

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, . . .

A proof in Coq/SSReflect with ML4PG help

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

File Edit Options Buffers Tools Coq Proof-General Holes Help

```
Lemma fact_tail_aux_lemma :
  forall (a n : nat), fact_tail_aux n a = a * n`!.
Proof.
move => n. elim : n => [a| n IH a /=].
  by rewrite /theta_fact fact0 muln1.
```

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

```
n : nat
IH : forall a : nat,
  fact_tail_aux n a = a * n`!
a : nat
================================
fact_tail_aux n (n.+1 * a) =
    a * (n.+1)`!
```

```
This lemma is similar to lemmas:
 - mult_tail_aux_lemma
 - power_tail_aux_lemma
 - expt_tail_aux_lemma
```

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

Where is our tool useful?

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

Where is our tool useful?

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

Suggestions for fn_fact_is_theta :
fn_expt_is_theta , fn_mult_is_theta , fn_power_is_theta

Where is our tool useful?

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

Suggestions for program_is_fn_fact :
program_is_fn_expt , program_is_fn_mult , program_is_fn_power

Where is our tool useful?

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

Suggestions for total_correctness_fact :
total_correctness_expt , total_correctness_mult , total_correctness_power

Benefits of this approach:

- ML4PG statistical tool can be switched on/off on user's demand;

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;

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;

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.

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.

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...

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!!!!

Table of Contents

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
3. More Examples
4. 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.
```

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

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

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

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

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

Proof

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

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

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

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

Proof

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

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

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

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

Proof

$$\begin{aligned} \sum_{1 \le i \le k} \sum_{l < j \le m} (\beta_n^{j,i-1} - \beta_n^{j,i}) - (\beta_n^{j-1,i-1} - \beta_n^{j-1,i}) &= \\ \sum_{1 \le i \le k} (\beta_n^{m,i-1} - \beta_n^{m,i}) - (\beta_n^{l,i-1} - \beta_n^{l,i}) = \dots \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

Suggestions provided by ML4PG

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

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

Suggestions provided by ML4PG

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

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

Suggestions provided by ML4PG

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

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

Lemma (Another ML4PG suggestion)

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

The CoqEAL library

M. Dénès and A. Mörtberg and V. Siles. A refinement-based approach to computational algebra in Coq. In: Proceedings Interactive Theorem Proving 2012 (ITP 2012). Lecture Notes in Computer Science 7406, 83–98. 2012.

The CoqEAL library

M. Dénès and A. Mörtberg and V. Siles. A refinement-based approach to computational algebra in Coq. In: Proceedings Interactive Theorem Proving 2012 (ITP 2012). Lecture Notes in Computer Science 7406, 83–98. 2012.

A methodology, based on the notion of refinement to formalise efficient algorithms of Computer Algebra systems:

The CoqEAL library

M. Dénès and A. Mörtberg and V. Siles. A refinement-based approach to computational algebra in Coq. In: Proceedings Interactive Theorem Proving 2012 (ITP 2012). Lecture Notes in Computer Science 7406, 83–98. 2012.

A methodology, based on the notion of refinement to formalise efficient algorithms of Computer Algebra systems:

1. Define the algorithm relying on rich dependent types

The CoqEAL library

M. Dénès and A. Mörtberg and V. Siles. A refinement-based approach to computational algebra in Coq. In: Proceedings Interactive Theorem Proving 2012 (ITP 2012). Lecture Notes in Computer Science 7406, 83–98. 2012.

A methodology, based on the notion of refinement to formalise efficient algorithms of Computer Algebra systems:

1. Define the algorithm relying on rich dependent types
2. Refine it to an efficient version described on high-level data structures

The CoqEAL library

M. Dénès and A. Mörtberg and V. Siles. A refinement-based approach to computational algebra in Coq. In: Proceedings Interactive Theorem Proving 2012 (ITP 2012). Lecture Notes in Computer Science 7406, 83–98. 2012.

A methodology, based on the notion of refinement to formalise efficient algorithms of Computer Algebra systems:

1. Define the algorithm relying on rich dependent types
2. Refine it to an efficient version described on high-level data structures
3. Implement it on data structures closer to machine representations

The CoqEAL library

M. Dénès and A. Mörtberg and V. Siles. A refinement-based approach to computational algebra in Coq. In: Proceedings Interactive Theorem Proving 2012 (ITP 2012). Lecture Notes in Computer Science 7406, 83–98. 2012.

A methodology, based on the notion of refinement to formalise efficient algorithms of Computer Algebra systems:

1. Define the algorithm relying on rich dependent types
2. Refine it to an efficient version described on high-level data structures
3. Implement it on data structures closer to machine representations

Problem

Decipher the key results which can help us to solve our concrete problems

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Problems:

- Prove the equivalence with the invmx algorithm of SSReflect

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Problems:

- Prove the equivalence with the invmx algorithm of SSReflect
- Executability of the algorithm

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Problems:

- Prove the equivalence with the invmx algorithm of SSReflect
- Executability of the algorithm

Suggestions:

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Problems:

- Prove the equivalence with the invmx algorithm of SSReflect
- Executability of the algorithm

Suggestions:

- Clustering with matrix library of SSReflect and CoqEAL library (˜ 1000)
- 10 suggestions
- Instead of proving:

```
Lemma fast_invmxE : forall m (M : 'M[R]_m), lower1 M ->
        fast_invmx M = invmx M.
```

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Problems:

- Prove the equivalence with the invmx algorithm of SSReflect
- Executability of the algorithm

Suggestions:

- Clustering with matrix library of SSReflect and CoqEAL library (˜ 1000)
- 10 suggestions
- Prove:

```
Lemma fast_invmxE : forall m (M : 'M[R]_m), lower1 M ->
       M *m fast_invmx M = 1%:M.
```

- Key suggestion:

```
Lemma invmx_is_uniq : forall m (M1 M2 : 'M[R]_m), M1 *m M2 = 1%:M ->
     M2 = invmx M1.
```

Fast inverse for triangular matrices

Suppose that we have defined a fast algorithm to compute the inverse of triangular matrices over a field called fast_invmx

Problems:

- Prove the equivalence with the invmx algorithm of SSReflect
- Executability of the algorithm

Suggestions:

- CoqEAL suggestion: refine the algorithm to work with sequences instead of matrices
- Clustering with CoqEAL library (˜ 700)
- 7 suggestions all of them related to the refinement from matrices to sequences

Table of Contents

1. Motivation: machine-learning for automated theorem proving?
2. Two main trends: ATP and ITP
3. More Examples
4. Conclusions and Further work

Conclusions and Further work

- 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.
- I would personally avoid “brute-force” methods for feature extraction, and would generally prefer an adaptable, perhaps genetic, algorithms for this purpose.
- Conceptualisation of ML4PG output is a challenge.

Conclusions and Further work

- 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.
- I would personally avoid “brute-force” methods for feature extraction, and would generally prefer an adaptable, perhaps genetic, algorithms for this purpose.
- Conceptualisation of ML4PG output is a challenge.

Dissemination

- Industrial applications: software and hardware verification (Centaur Technology, Rockwell Collins)
- Among peers (researchers, mathematicians, programmers)?

Can statistical machine learning advance mechanised proof technology?

Katya Komendantskaya, joint work with Jonathan Heras
(Funded by EPSRC First Grant Scheme)

University of Dundee

19 June 2013