Machine Learning for Proof General: Interfacing Interfaces
(Funded by EPSRC First Grant Scheme)

Katya Komendantskaya and Jonathan Heras

University of Dundee

30 November 2012

Outline

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

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Automation of interactive proofs: role of interfaces...

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Automation of interactive proofs: role of interfaces...
3. ML4PG: machine-learning for proof general

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Automation of interactive proofs: role of interfaces...
3. ML4PG: machine-learning for proof general
4. Amazing Examples
   - The bigop library
   - The CoqEAL library
   - Formalisation of the Java Virtual Machine

Outline

1. Motivation: machine-learning for automated theorem proving?
2. Automation of interactive proofs: role of interfaces...
3. ML4PG: machine-learning for proof general
4. Amazing Examples
   - The bigop library
   - The CoqEAL library
   - Formalisation of the Java Virtual Machine
5. Further work

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 infeasible to be processed and interpreted manually.

Why Machine-Learning?

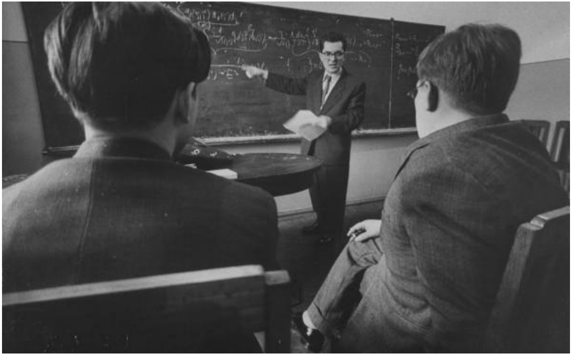
- ... Digital era means most of information (in science, industries, even art!) is stored/handled in electronic form.
- ... Computer-generated data may not make much sense to human users; or in fact, other computers!
- The volumes of data make it infeasible to be processed and interpreted manually.
-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

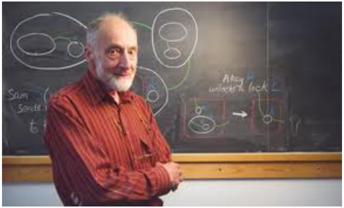
So, why should we (logicians) care?



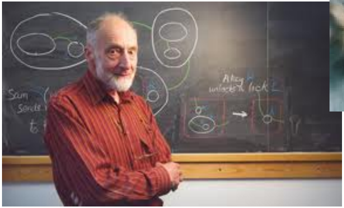
The answer is...

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 ITPs 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 ITPs 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 appliences, 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-throughs (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.

So,...

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

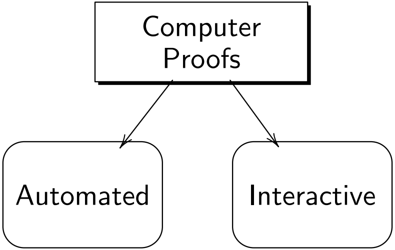
- finding common proof-patterns in proofs across various scripts, libraries, users, notations;
- providing proof-hints, especially in (industrial) cases where routine similar cases are frequent, and proof development is distributed across several programmers.

Patty and I are considering using machine-learning to generate hints in undecidable cases of Higher-order unification.

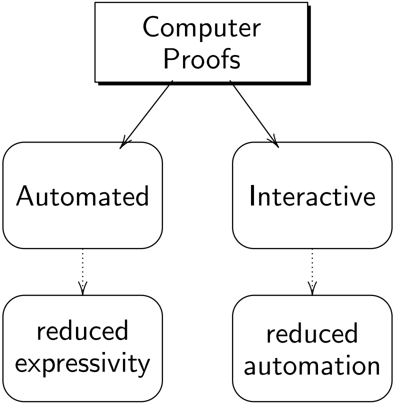
Outline

1. Motivation: machine-learning for automated theorem proving?
2. Automation of interactive proofs: role of interfaces...
3. ML4PG: machine-learning for proof general
4. Amazing Examples
   - The bigop library
   - The CoqEAL library
   - Formalisation of the Java Virtual Machine
5. Further work

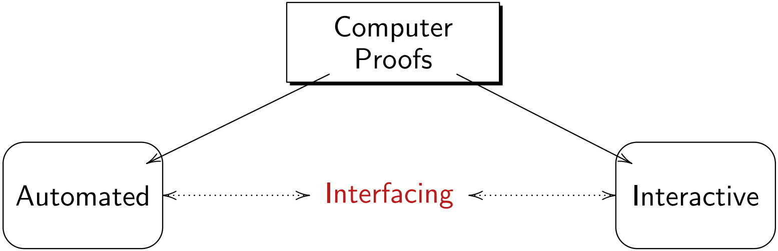
Interfacing-1:



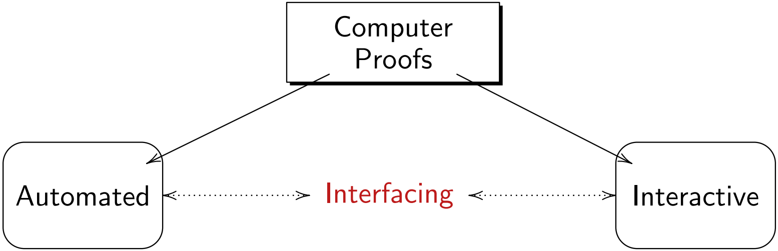
Interfacing-1:



Solution? – Interfacing

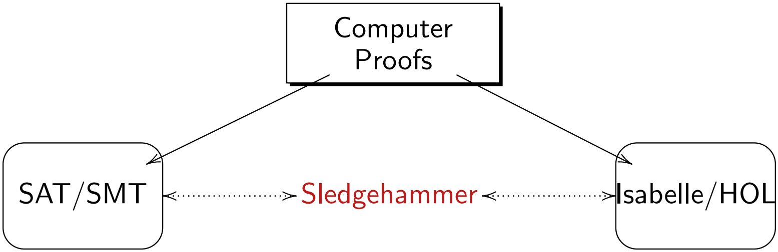


Solution? – Interfacing

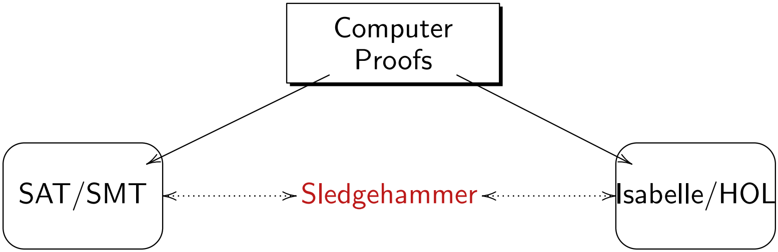


ITP environment allows the user to “call” ATP for generating solutions.

Solution? – Interfacing. Example

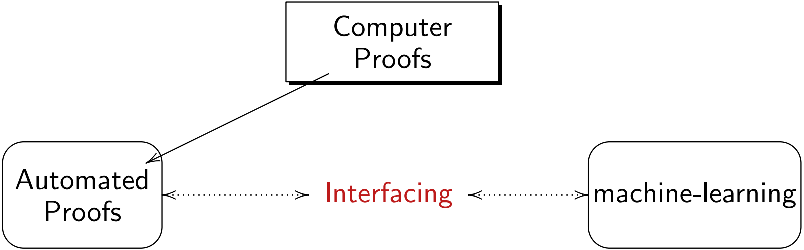


Solution? – Interfacing. Example

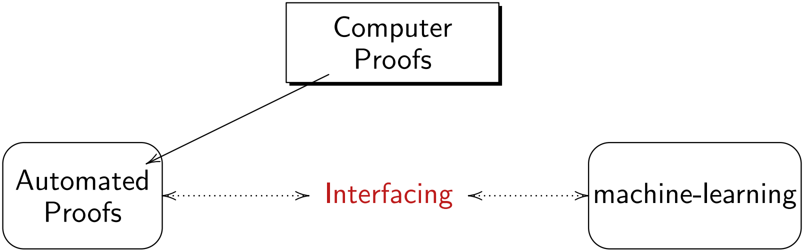


A note: forward interfacing is easier than backwards interfacing.

Less familiar alternative:

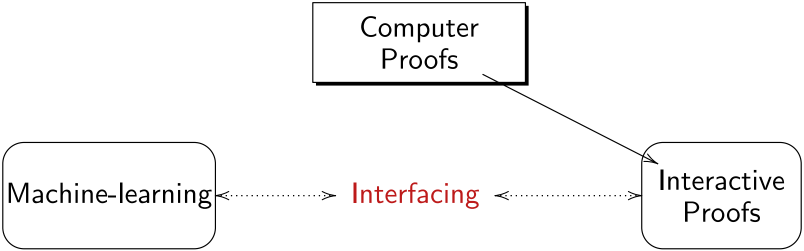


Less familiar alternative:

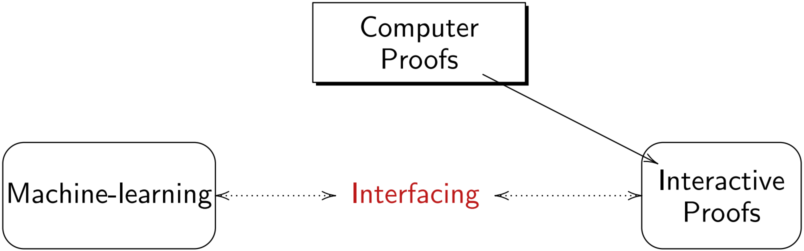


Benefits: learning “proof heuristics”, speed up in computations.
Some success: e.g. work by Stephan Shulz, Joseph Urban.

Less familiar alternative:



Less familiar alternative:



Benefits: helping users to handle big proof developments and libraries. Some attempts: Alan Bundy and Hazel Duncan, current AI4FM project (Edinburgh and Newcastle).

Why machine-learning interactive proofs is harder?

- The richer language reduces the chance of finding regularities and proof patterns by data-mining the syntax alone. Moreover, in ITPs, one and the same goal may have a range of different proofs, whereas different goals can be proven by the same sequence of tactics.

Why machine-learning interactive proofs is harder?

- The richer language reduces the chance of finding regularities and proof patterns by data-mining the syntax alone. Moreover, in ITPs, one and the same goal may have a range of different proofs, whereas different goals can be proven by the same sequence of tactics.
- The notions of a *proof* may be regarded from different perspectives in ITPs: it may be seen as a transition between the subgoals, a combination of applied tactics, or — more traditionally – a proof-tree showing the overall proof strategy.

Why machine-learning interactive proofs is harder?

- The richer language reduces the chance of finding regularities and proof patterns by data-mining the syntax alone. Moreover, in ITPs, one and the same goal may have a range of different proofs, whereas different goals can be proven by the same sequence of tactics.
- The notions of a *proof* may be regarded from different perspectives in ITPs: it may be seen as a transition between the subgoals, a combination of applied tactics, or — more traditionally – a proof-tree showing the overall proof strategy.

Demo...

$$\sum_{1}^{n} i = \frac{n(n+1)}{2}$$

Our solution?

Machine-learning tools for ITPs need to be interactive themsleves:

they should guide the user.

Our solution?

Machine-learning tools for ITPs need to be interactive themsleves:

they should guide the user.

– start with new user interface:

... new machine-learning extension of Proof General (itself an interface for a variety of ITPs).

Our solution?

Machine-learning tools for ITPs need to be interactive themsleves:

they should guide the user.

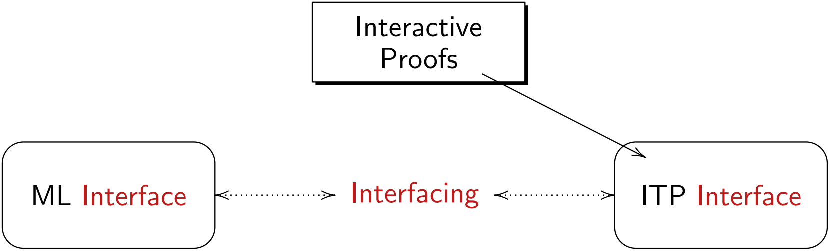
– start with new user interface:

... new machine-learning extension of Proof General (itself an interface for a variety of ITPs).

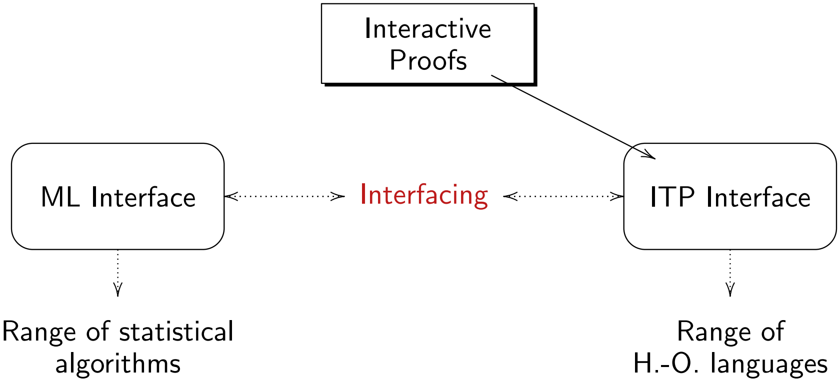
Note: – similarly –

– huge role user interfaces play in Machine-learning community: MATLAB, WEKA, – are famous interfaces to run a range of statistical algorithms.

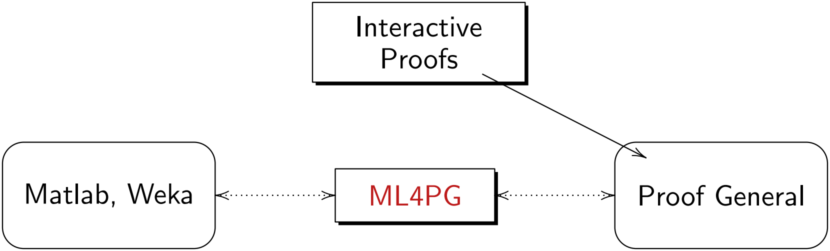
Our solution: Interfacing Interfaces:



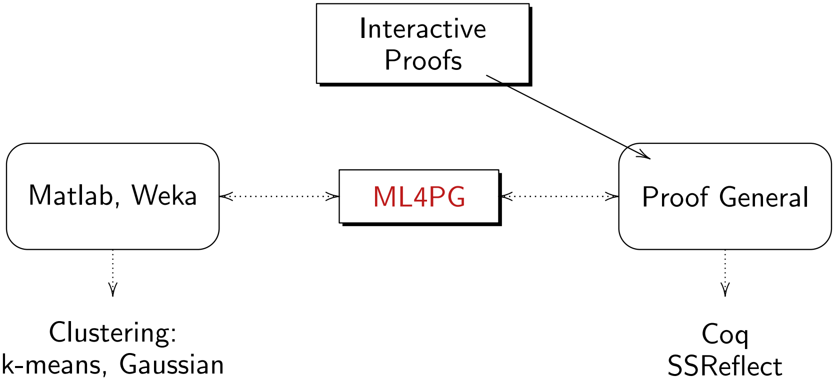
Our solution: Interfacing Interfaces:



Our solution: ML4PG:

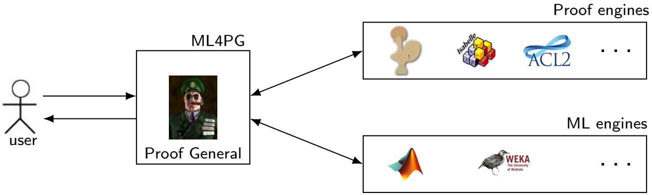


Our solution: ML4PG:

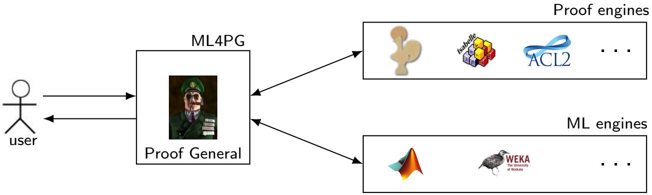


Outline

Overall architecture of ML4PG

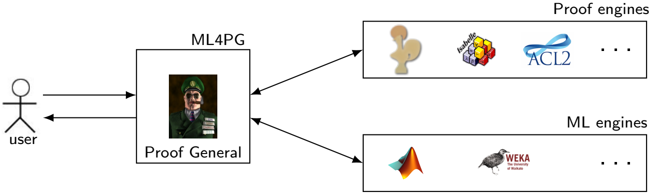


Overall architecture of ML4PG



Interaction with ML4PG:

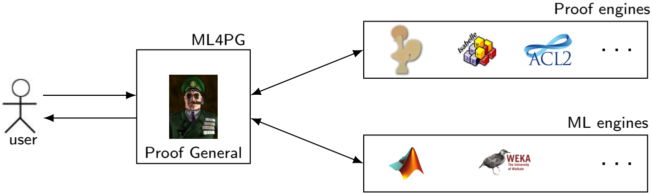
Overall architecture of ML4PG



Interaction with ML4PG:

- User interacts with Proof General as usual,

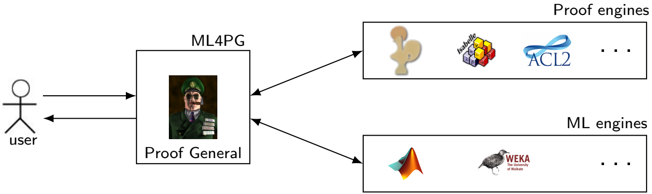
Overall architecture of ML4PG



Interaction with ML4PG:

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

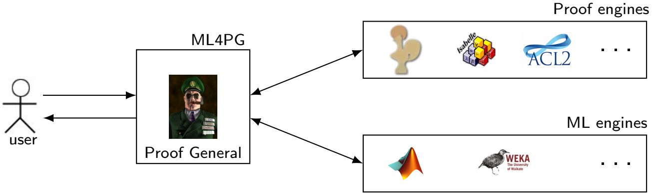
Overall architecture of ML4PG



Interaction with ML4PG:

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

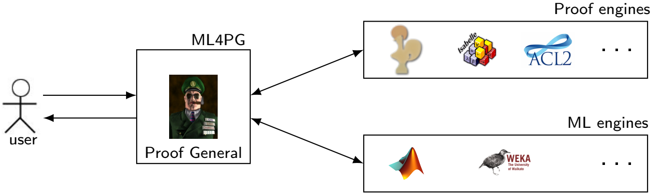
Overall architecture of ML4PG



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.

The most clever part... Feature extraction in ML4PG

Problem:

- statistical ML tools expect, as input, a fixed number of features describing all objects to be classified;

The most clever part... Feature extraction in ML4PG

Problem:

- statistical ML tools expect, as input, a fixed number of features describing all objects to be classified;
- in higher-order proofs, we cannot fix a finite number of goal shapes or proofs configurations to describe all possible proofs;

The most clever part... Feature extraction in ML4PG

Problem:

- statistical ML tools expect, as input, a fixed number of features describing all objects to be classified;
- in higher-order proofs, we cannot fix a finite number of goal shapes or proofs configurations to describe all possible proofs;
- we gather statistics based on a fixed number of implicit proof parameters – proof traces.

Solution: Proof Trace Method...

...gathers statistics on the basis of:

Solution: Proof Trace Method...

...gathers statistics on the basis of:

- ways the user treats the goal – i.e. which tactics he applies

Solution: Proof Trace Method...

...gathers statistics on the basis of:

- ways the user treats the goal – i.e. which tactics he applies
- simple parameters such as top symbol, types of arguments, number of generated subgoals...

Solution: Proof Trace Method...

...gathers statistics on the basis of:

- ways the user treats the goal – i.e. which tactics he applies
- simple parameters such as top symbol, types of arguments, number of generated subgoals...
- the relative transformation of these parameters within several proof-steps.

An "ordinary miracle":

- Neither of the parameters: tactic sequence, goal shape, or argument types is sufficient on its own for drawing conclusions about significant proof patterns;

Solution: Proof Trace Method...

...gathers statistics on the basis of:

- ways the user treats the goal – i.e. which tactics he applies
- simple parameters such as top symbol, types of arguments, number of generated subgoals...
- the relative transformation of these parameters within several proof-steps.

An "ordinary miracle":

- Neither of the parameters: tactic sequence, goal shape, or argument types is sufficient on its own for drawing conclusions about significant proof patterns;
- For one proof step, the collection of these parameters is insufficient for meaningful proof-pattern recognition;

Solution: Proof Trace Method...

...gathers statistics on the basis of:

- ways the user treats the goal – i.e. which tactics he applies
- simple parameters such as top symbol, types of arguments, number of generated subgoals...
- the relative transformation of these parameters within several proof-steps.

An "ordinary miracle":

- Neither of the parameters: tactic sequence, goal shape, or argument types is sufficient on its own for drawing conclusions about significant proof patterns;
- For one proof step, the collection of these parameters is insufficient for meaningful proof-pattern recognition;
- Collection of these features over several proof steps – a proof trace – gives amazing results.

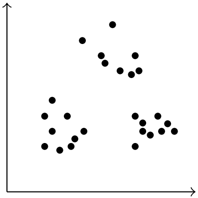
ML4PG approach to proof-clustering

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:

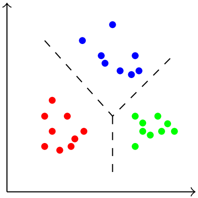
- Unsupervised machine learning technique:



ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

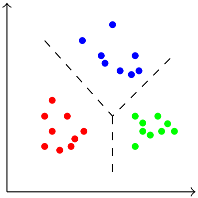


- Engines: Matlab, Weka, Octave, R, . . .

ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

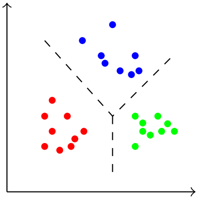


- Engines: Matlab, Weka, Octave, R, . . .

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, ...

ML4PG approach to proof-clustering

- Clustering algorithms:
- Problem:
  - Results of one run may differ from another, for one data set.

ML4PG approach to proof-clustering

- Clustering algorithms:
- Problem:
  - Results of one run may differ from another, for one data set.
- Solution:
  - Certain clusters are found repeatedly in different runs.
  - Certain clusters have higher “similarity” value than others (0.5 is our threshold).

ML4PG approach to proof-clustering

- Clustering algorithms:
- Problem:
  - Results of one run may differ from another, for one data set.
- Solution:
  - Certain clusters are found repeatedly in different runs.
  - Certain clusters have higher "similarity" value than others (0.5 is our threshold).
  - Those clusters are the most reliable.

ML4PG approach to proof-clustering

- Clustering algorithms:
- Problem:
  - Results of one run may differ from another, for one data set.
- Solution:
  - Certain clusters are found repeatedly in different runs.
  - Certain clusters have higher "similarity" value than others (0.5 is our threshold).
  - Those clusters are the most reliable.
- We have extended Matlab and Weka programs to obtain the most frequent clusters.

ML4PG approach to proof-clustering

- Clustering algorithms:
- Problem:
  - Results of one run may differ from another, for one data set.
- Solution:
  - Certain clusters are found repeatedly in different runs.
  - Certain clusters have higher "similarity" value than others (0.5 is our threshold).
  - Those clusters are the most reliable.
- We have extended Matlab and Weka programs to obtain the most frequent clusters.

  This means the ML4PG user does not have to analyse the statistics manually!!!

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: in our case – from Coq to SSReflect.

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: in our case – from Coq to SSReflect.

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: in our case – from Coq to SSReflect.

Most amazingly...

it really works!!!!

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: in our case – from Coq to SSReflect.

Most amazingly...

it really works!!!!

Demo...

Demo: ML4PG options and various clusters

$$\sum_{1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6},$$

$$\sum_{1}^{n} i^3 = \frac{n^4+2n^3+n^2}{4},$$

$$\sum_{1}^{n}(2i-1) = n^2,$$

$$\sum_{1}^{n}(2i-1)^2 = \frac{4n^3-n}{3},$$

$$\sum^{n}(2i-1)^3 = 2n^4-n^2.$$

Table of Contents

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:

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

- 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

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

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

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

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

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

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

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

Suggestions provided by ML4PG

Theorem (Fundamental Lemma of Persistent Homology)

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

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

Proof

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

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

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

Proof

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

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

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

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

Formalisation of the JVM: example suggested by J Moore

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

Formalisation of the JVM: example suggested by J Moore

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

Goal

- Model a subset of the JVM in Coq, defining an interpreter for JVM programs
- Verify the correctness of JVM programs within Coq

Formalisation of the JVM: example suggested by J Moore

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

Goal

- Model a subset of the JVM in 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 analytical 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.

An example: computing 5!

Java code:

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

An example: computing 5!

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*

An example: computing 5!

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				...

An example: computing 5!

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				...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

Bytecode:

. . .

JVM model:

. . .

An example: computing 5!

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			...

An example: computing 5!

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			...

An example: computing 5!

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			...

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`!).
```

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`

Table of Contents

Further work

- not only trace successful proofs, but also failed and discarded derivation steps;

Further work

- not only trace successful proofs, but also failed and discarded derivation steps;
- increase the number of Interactive Theorem Provers and Machine Learning engines;

Further work

- not only trace successful proofs, but also failed and discarded derivation steps;
- increase the number of Interactive Theorem Provers and Machine Learning engines;
- replace local environment with a client-server framework.

Job add

- University of Dundee is about to announce positions of Dundee Fellows;
- 5-year fellow position, becoming a permanent lectureship at the end; starts at 8 point scale;
- ITPs were selected as one of a few “named” areas;
- competition will be across several school and departments;
- if you know potential winner – please let me know.

Machine Learning for Proof General: Interfacing Interfaces (Funded by EPSRC First Grant Scheme)

Katya Komendantskaya and Jonathan Heras

University of Dundee

30 November 2012