Statistical Machine Learning in Interactive Theorem Proving

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

University of Dundee

18 July 2014

Outline

Outline

1. Existing tools and challenges
2. ML4PG: "Machine Learning for Proof General"

Outline

Outline

Challenges for ITP users:

- . . . size and sophistication of libraries stand on the way of efficient knowledge reuse;
  - . . . 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 for ITP users:

- . . . size and sophistication of libraries stand on the way of efficient knowledge reuse;
  - . . . 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.
- . . . manual handling of various proofs, strategies, libraries, becomes difficult;

Challenges for ITP users:

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

Challenges for ITP users:

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

Existing tools for managing Coq libraries: Search

Several searching tools in Coq:

- `Search` , `SearchAbout`, `SearchPattern` and `SearchRewrite`.
- SSReflect implements its own version `Search` with functionality the 4 Coq's search commands.

Example

```
Search "distr"in bigop
Search _ (_ * (\big[_/_]_(_ <- _| _)_))
```

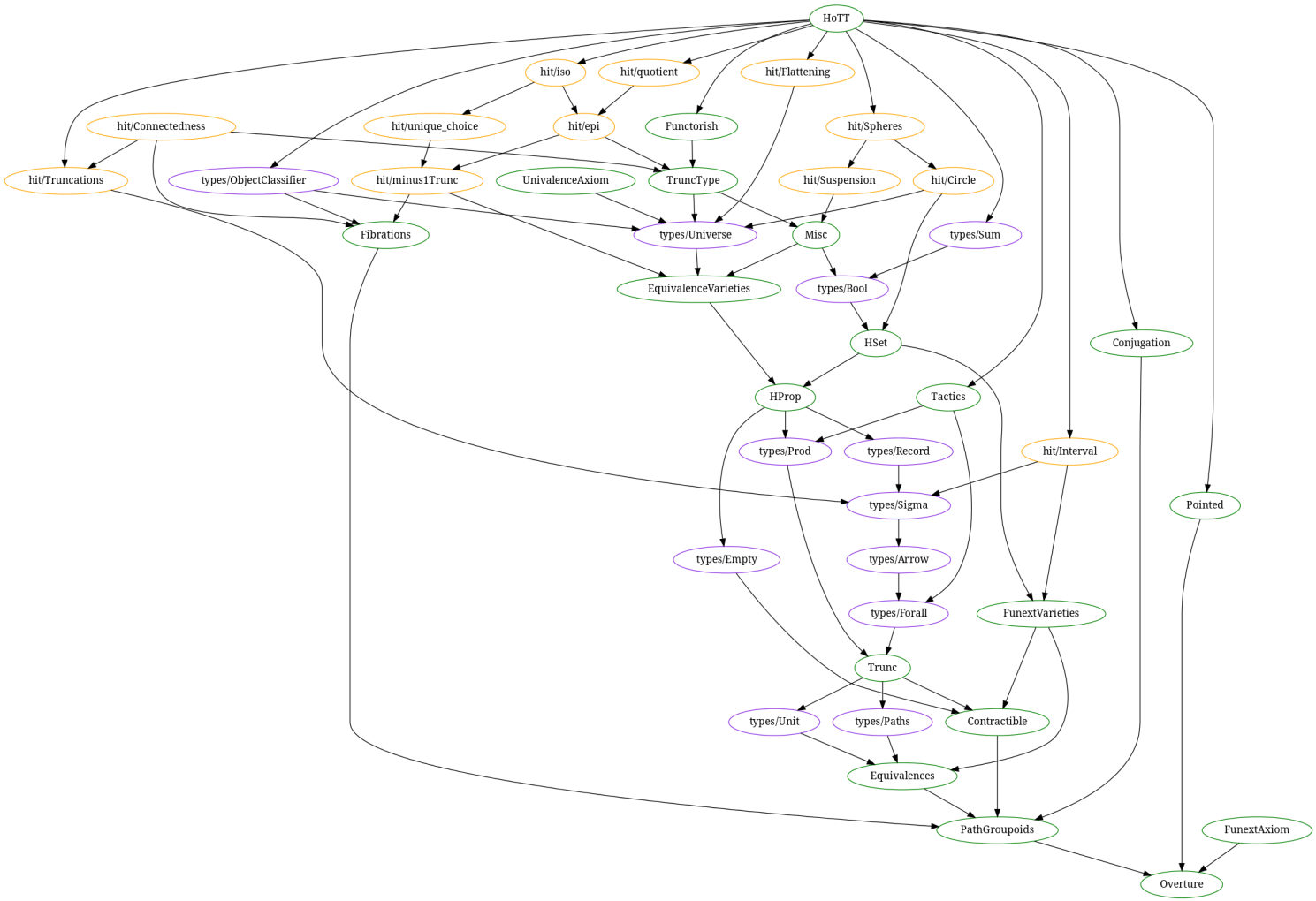
- The Whelp platform is a web search engine in Coq code, whith 3 functions:
  `Match` (similar `Search`),
  `Hint` (finds all the theorems which can be applied to derive the current goal) and
  `Elim` (retrieves all the eliminators of a given type).

Main properties of the search engines:

- goal oriented
- hence the user should already know what he is searching for: pattern/library/lemma name, etc
- deterministic: if the exact requested pattern exists, they will find it.

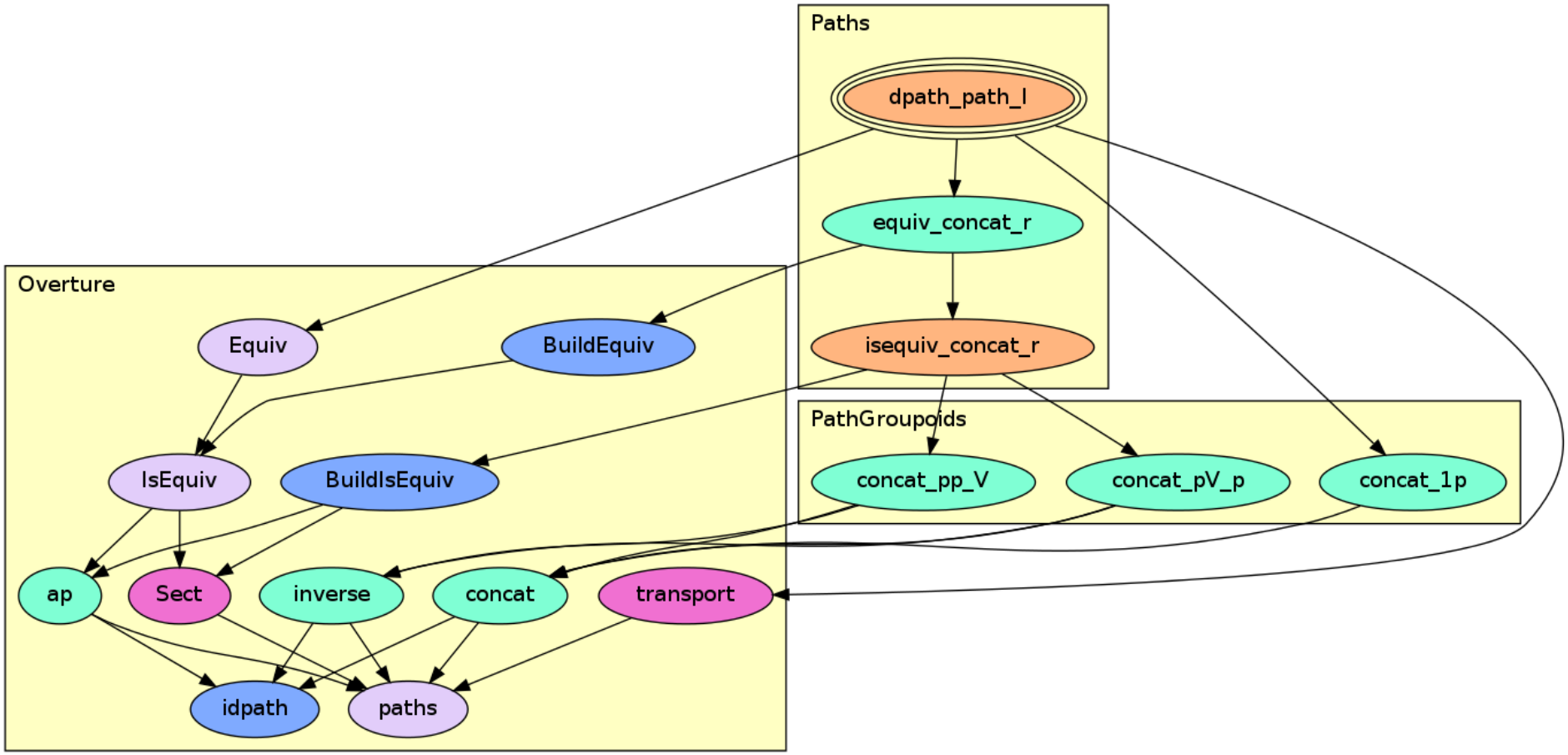
Existing tools-2: Dependency graphs

- You do not have to know what you are searching for
- They show "all there is".



Existing tools-2: Dependency graphs

- or perhaps all there is relative to your lemma/term



Dependency graphs DOs and DON'Ts

- nicely visualised
- not goal directed – but can be used for a goal
- deterministic: if there is a dependency, it will be shown
- but it would not tell you if there are similar lemmas/terms
- it would not tell you which of those dependencies are more important than others for the proof
- there may be excessive information that actually hides the essence of the proof

The missing tool...

Something that could help us to:

- capture and search the meaning of the libraries;
- the higher-level proof strategies beyond tactics and notations
- identify redundancies and repetitions....

Motivating example:

Theorem (Fundamental Lemma of Persistent Homology)

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

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

Apply case on n.

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

Same strategy:

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

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

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

Lemma

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

Outline

Proof pattern recognition in ITPs

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;
- provide proof-hints in real-time and relative to a proof stage;
- assist the user, not the prover.

Proof pattern recognition in ITPs

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

Apply machine-learning methods to:

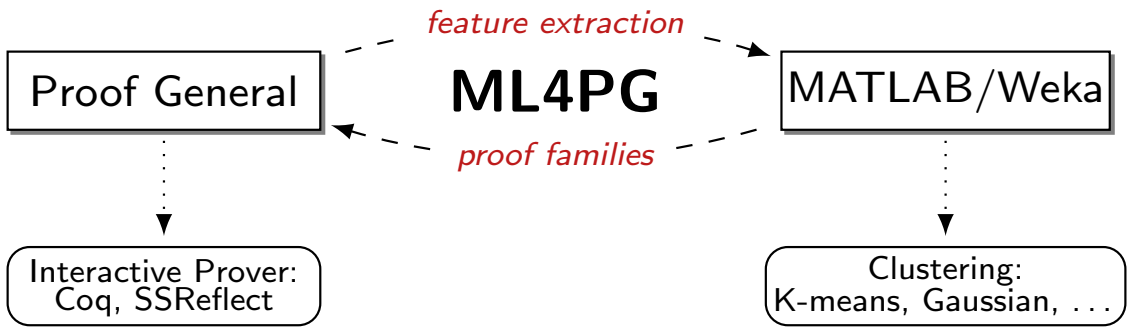
- find common proof-patterns in proofs across various scripts, libraries, users and notations;
- provide proof-hints in real-time and relative to a proof stage;
- assist the user, not the prover.

ML4PG:

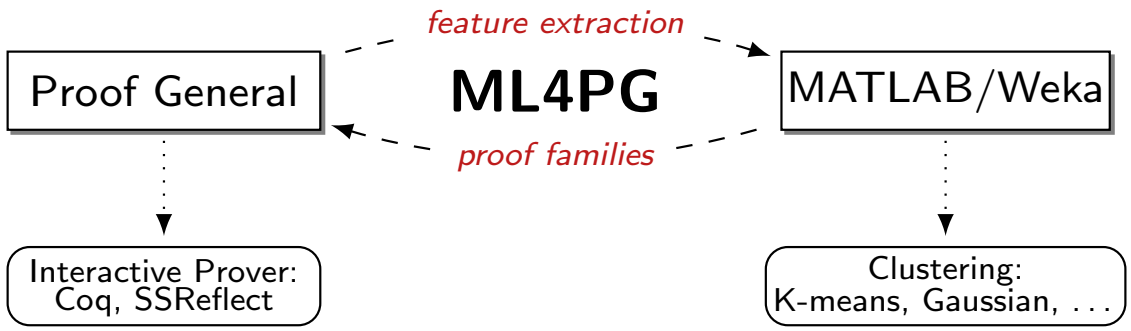
- Proof General extension which applies machine learning methods to Coq/SSReflect proofs. [Now available in standard Proof General distribution]

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

Machine Learning 4 Proof General: interfacing interfaces



Machine Learning 4 Proof General: interfacing interfaces



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

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

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

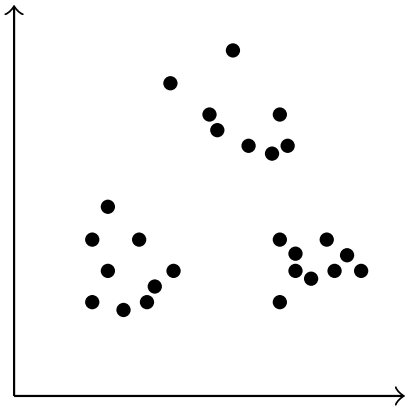
ML4PG approach to proof-clustering

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:

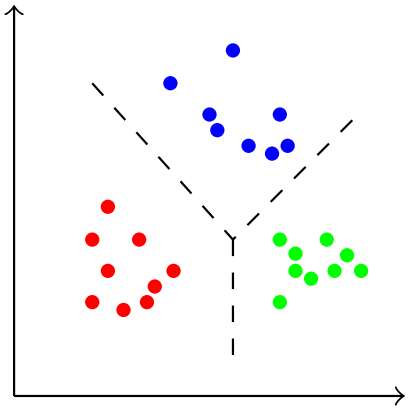
- Unsupervised machine learning technique:



ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

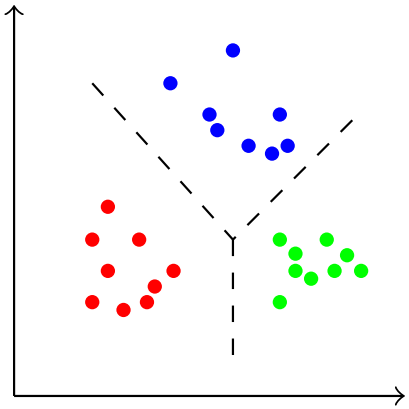


- Engines: Matlab, Weka, Octave, R, ...

ML4PG approach to proof-clustering

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

- Unsupervised machine learning technique:

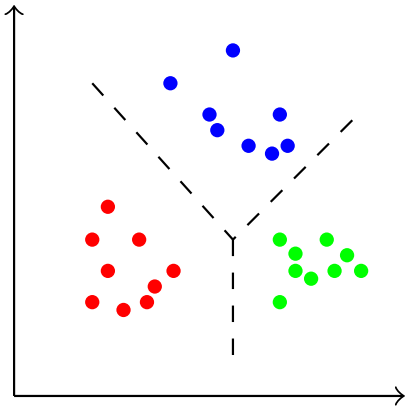


- Engines: Matlab, Weka, Octave, R, ...

ML4PG approach to proof-clustering

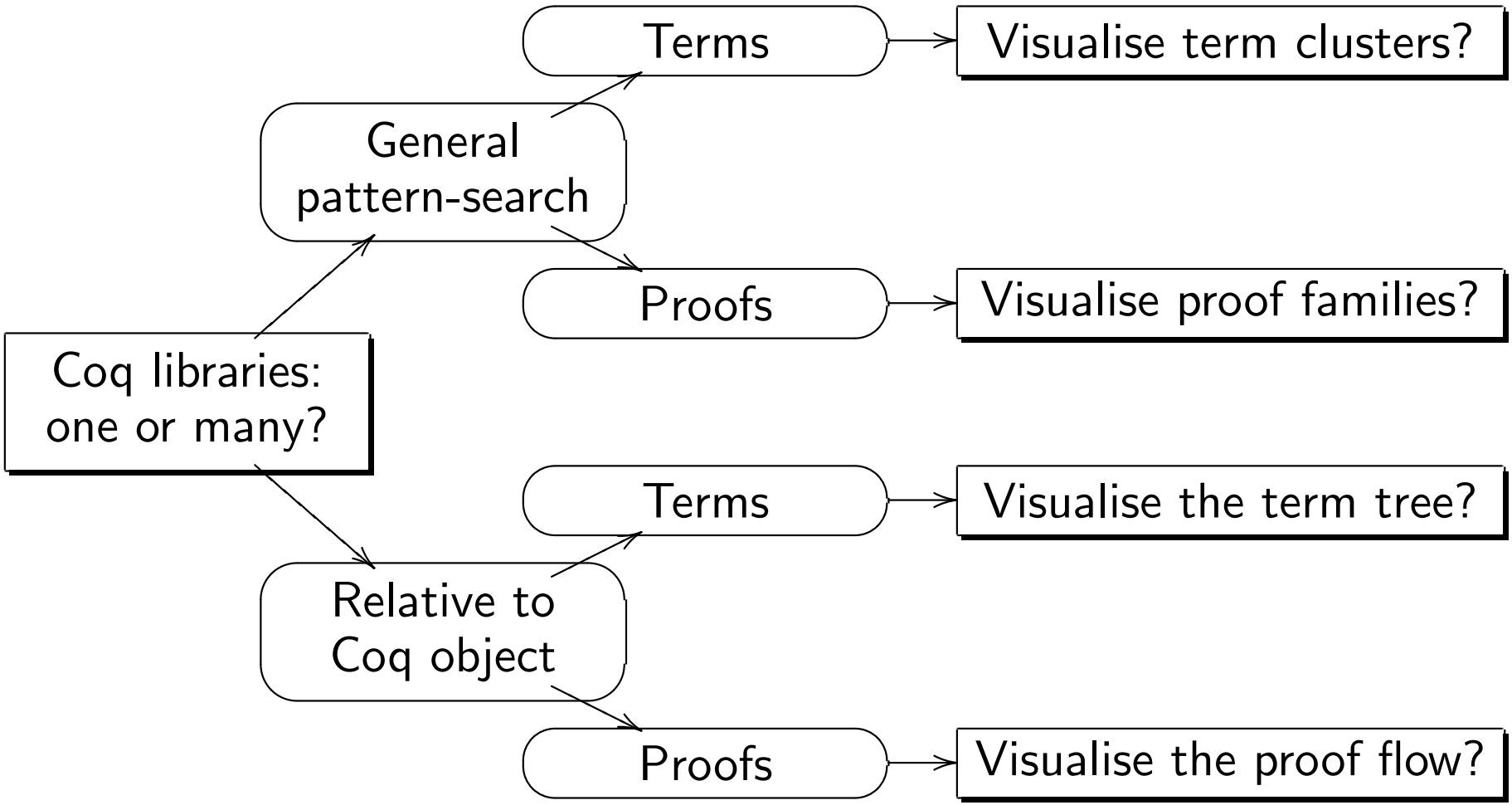
We have integrated Proof General with a variety of clustering algorithms:

- Unsupervised machine learning technique:

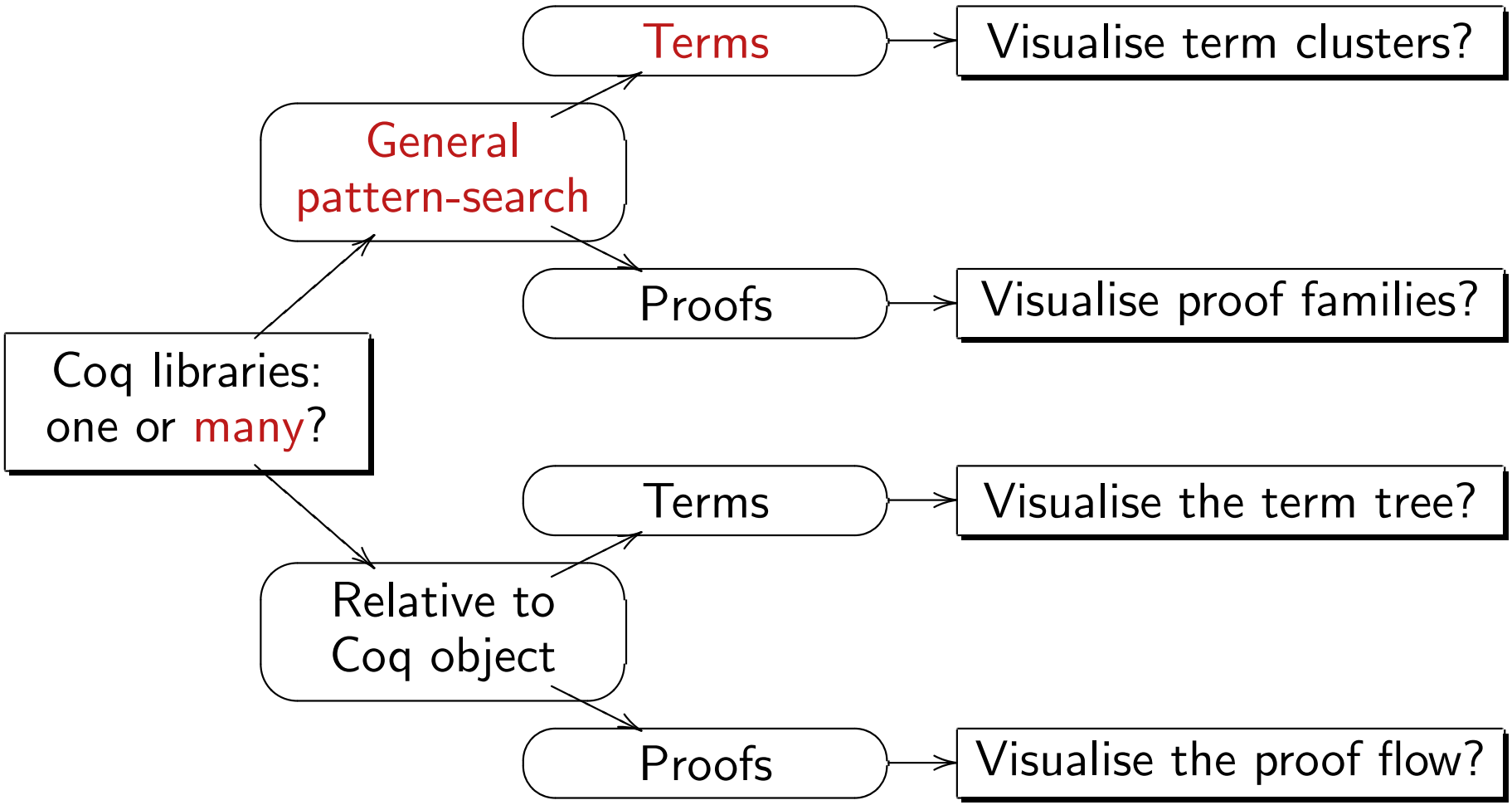


- Engines: Matlab, Weka, Octave, R, ...
- Algorithms: K-means, Gaussian Mixture models, simple Expectation Maximisation, ...

Feature extraction and output strategies



Feature extraction



Feature Extraction for Terms

- specifically developed for Coq and ML4PG

Terms are understood broadly: ...Definitions, type declarations, (co)fixpoint function definitions, lemma and theorem statements...

Example

```
forall (n :  nat) (H : even n), odd (n + 1).
```

Feature Extraction for Terms

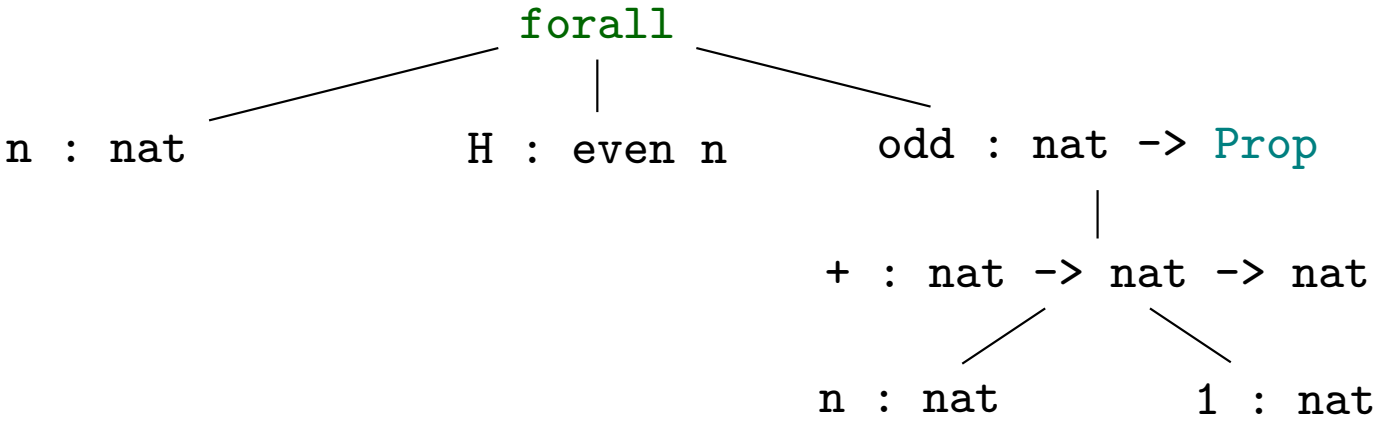
- specifically developed for Coq and ML4PG

Terms are understood broadly: ...Definitions, type declarations, (co)fixpoint function definitions, lemma and theorem statements...

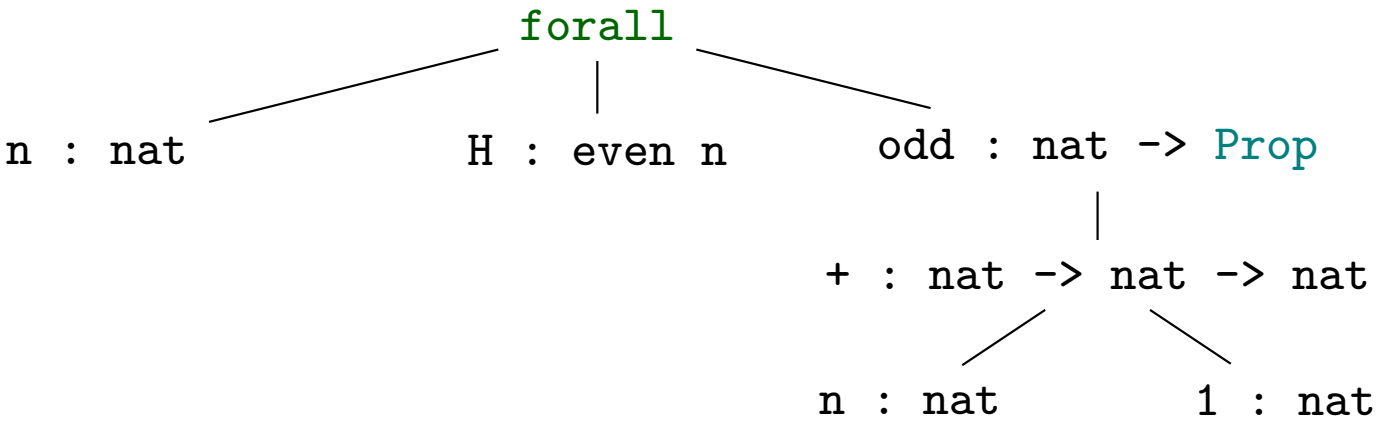
Example

```
forall (n :  nat) (H : even n), odd (n + 1).
```

Convert to a term-tree:

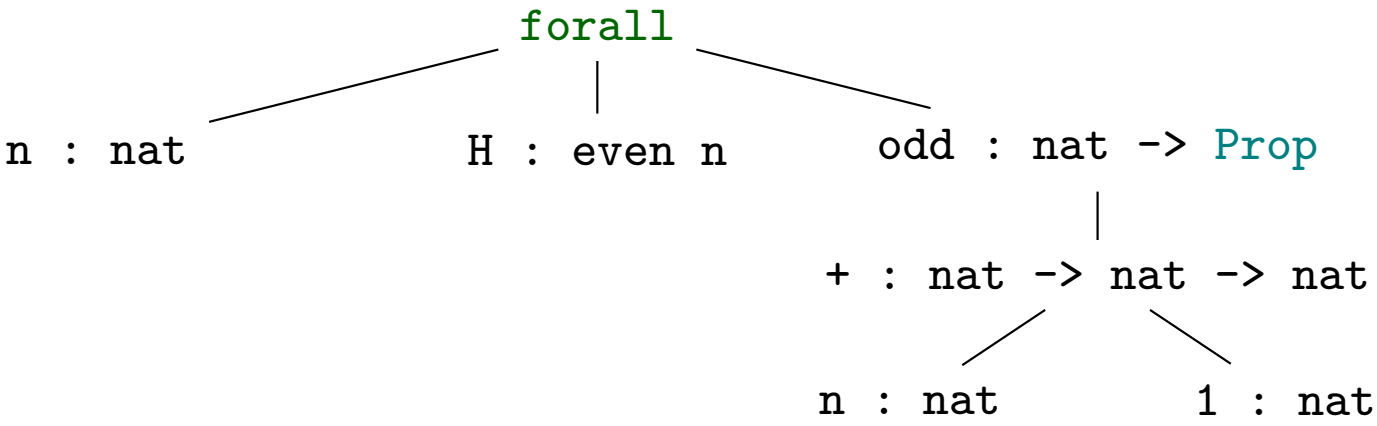


What are the important features of a term tree?



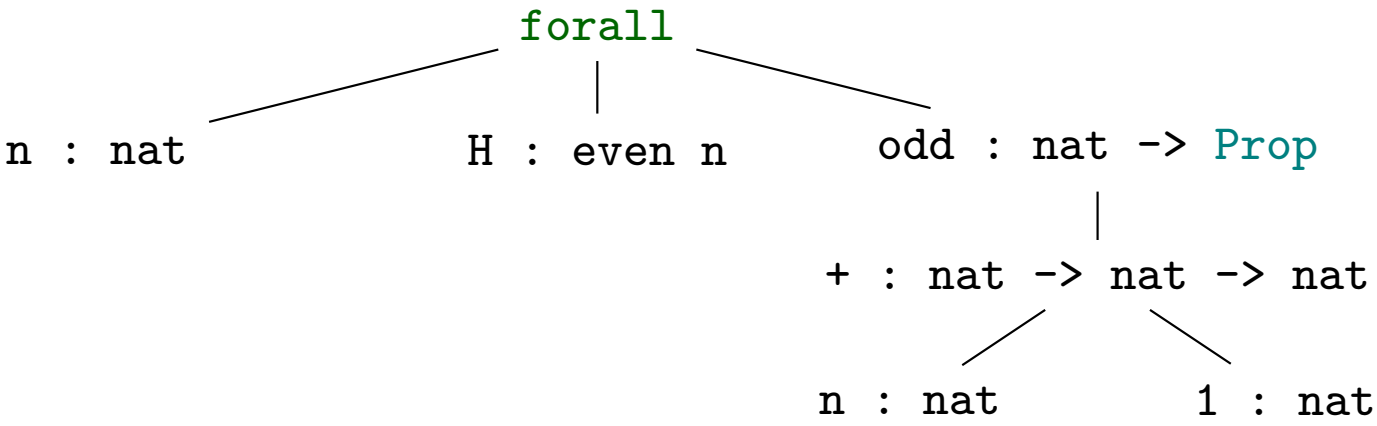
1. its topology

What are the important features of a term tree?



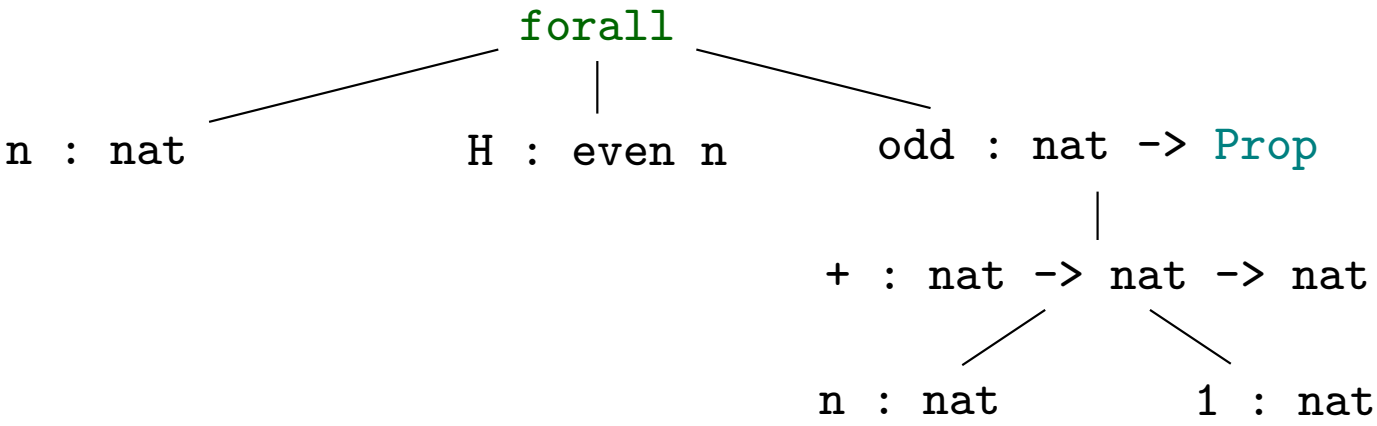
1. its topology
2. what populates its nodes:

What are the important features of a term tree?



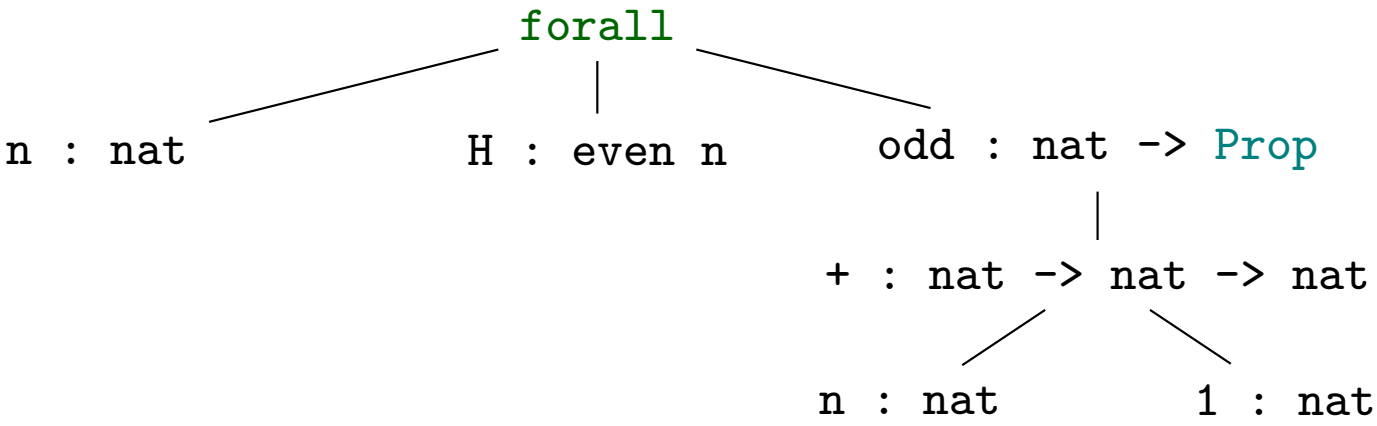
1. its topology
2. what populates its nodes:
   - terms and their types

What are the important features of a term tree?



1. its topology
2. what populates its nodes:
   - terms and their types
   - definitions of those terms and types, their meaning and structure...

What are the important features of a term tree?



1. its topology
2. what populates its nodes:
   - terms and their types
   - definitions of those terms and types, their meaning and structure...
   - their role in this proof library...

The method of Recurrent Clustering:

	level index 0	level index 1	level index 2
td0	([forall]$_{Gallina}$,-1,-1)	(0,0,0)	(0,0,0)
td1	([n]$_{term}$,[nat]$_{type}$,0)	([H]$_{term}$,[even n]$_{type}$,0)	([odd]$_{term}$,[nat->Prop]$_{type}$,0)
td2	([+]$_{term}$,[nat->nat->nat]$_{type}$,2)	(0,0,0)	(0,0,0)
td3	([n]$_{term}$,[nat]$_{type}$,0)	([1]$_{term}$,[nat]$_{type}$,0)	(0,0,0)

The method of Recurrent Clustering:

	level index 0	level index 1	level index 2
td0	([forall]$_{Gallina}$,-1,-1)	(0,0,0)	(0,0,0)
td1	([n]$_{term}$,[nat]$_{type}$,0)	([H]$_{term}$,[even n]$_{type}$,0)	([odd]$_{term}$,[nat->Prop]$_{type}$,0)
td2	([+]$_{term}$,[nat->nat->nat]$_{type}$,2)	(0,0,0)	(0,0,0)
td3	([n]$_{term}$,[nat]$_{type}$,0)	([1]$_{term}$,[nat]$_{type}$,0)	(0,0,0)

- The matrix captures the topology

The method of Recurrent Clustering:

	level index 0	level index 1	level index 2
td0	([forall]$_{Gallina}$,-1,-1)	(0,0,0)	(0,0,0)
td1	([n]$_{term}$,[nat]$_{type}$,0)	([H]$_{term}$,[even n]$_{type}$,0)	([odd]$_{term}$,[nat->Prop]$_{type}$,0)
td2	([+]$_{term}$,[nat->nat->nat]$_{type}$,2)	(0,0,0)	(0,0,0)
td3	([n]$_{term}$,[nat]$_{type}$,0)	([1]$_{term}$,[nat]$_{type}$,0)	(0,0,0)

- The matrix captures the topology
- The function `[.]` gives the meaning:

The method of Recurrent Clustering:

	level index 0	level index 1	level index 2
td0	([forall]$_{Gallina}$,-1,-1)	(0,0,0)	(0,0,0)
td1	([n]$_{term}$,[nat]$_{type}$,0)	([H]$_{term}$,[even n]$_{type}$,0)	([odd]$_{term}$,[nat->Prop]$_{type}$,0)
td2	([+]$_{term}$,[nat->nat->nat]$_{type}$,2)	(0,0,0)	(0,0,0)
td3	([n]$_{term}$,[nat]$_{type}$,0)	([1]$_{term}$,[nat]$_{type}$,0)	(0,0,0)

- The matrix captures the topology
- The function [.] gives the meaning:
- it is defined recurrently and adaptively, using clustering, for every given library and proof-stage;

The method of Recurrent Clustering:

	level index 0	level index 1	level index 2
td0	([forall]$_{Gallina}$,-1,-1)	(0,0,0)	(0,0,0)
td1	([n]$_{term}$,[nat]$_{type}$,0)	([H]$_{term}$,[even n]$_{type}$,0)	([odd]$_{term}$,[nat->Prop]$_{type}$,0)
td2	([+]$_{term}$,[nat->nat->nat]$_{type}$,2)	(0,0,0)	(0,0,0)
td3	([n]$_{term}$,[nat]$_{type}$,0)	([1]$_{term}$,[nat]$_{type}$,0)	(0,0,0)

- The matrix captures the topology
- The function [.] gives the meaning:
- it is defined recurrently and adaptively, using clustering, for every given library and proof-stage;
- ...starts with Gallina pre-defined symbols, and uses them to find similarity of the first few Coq definitions; and then proceeds recursively.

The method of Recurrent Clustering:

	level index 0	level index 1	level index 2
td0	([forall]$_{Gallina}$,-1,-1)	(0,0,0)	(0,0,0)
td1	([n]$_{term}$,[nat]$_{type}$,0)	([H]$_{term}$,[even n]$_{type}$,0)	([odd]$_{term}$,[nat->Prop]$_{type}$,0)
td2	([+]$_{term}$,[nat->nat->nat]$_{type}$,2)	(0,0,0)	(0,0,0)
td3	([n]$_{term}$,[nat]$_{type}$,0)	([1]$_{term}$,[nat]$_{type}$,0)	(0,0,0)

- The matrix captures the topology
- The function `[.]` gives the meaning:
- it is defined recurrently and adaptively, using clustering, for every given library and proof-stage;
- ...starts with Gallina pre-defined symbols, and uses them to find similarity of the first few Coq definitions; and then proceeds recursively.
- The more two terms or types are "semantically similar", the closer values they get. Thus, this matrix should have similar content to e.g. the matrix of `forall (n : nat) (H : odd n), even (n + 1).`

Motivation for this feature extraction:

For each term, we get a matrix of size up to 300 features, which capture:

1. term-tree structure of that term – via the term-depth, level-size, and additional "third" feature relating the above;
2. its types as related to terms; (pattern-recognition tools analyse the relative values of all features, as 1 Coq object is a point in 300-dimensional space)
3. its dependency to other definitions and Coq terms types – via recurrent clustering

A simple example

General library clustering:

SSReflect Base library, 12 standard files, 457 terms, 91 clusters (the number and size of clusters can be changed using PG options); 5-10 seconds.

Example

```
Fixpoint eqn (m n :  nat) :=
match m, n with
| 0, 0 => true
| m'.+1, n'.+1 => eqn m' n'
| _, _ => false end.
Fixpoint eqseq (s1 s2 :  seq T) :=
match s1, s2 with
| [::], [::]  => true
| x1 ::  s1', x2 ::  s2' => (x1 == x2) && eqseq s1' s2'
| _, _ => false end.
```

Note: common structure across types and type constructors

A simple example

General library clustering:

SSReflect Base library, 12 standard files, 457 terms, 91 clusters (the number and size of clusters can be changed using PG options); 5-10 seconds.

Example

```
Fixpoint drop n s := match s, n with
 | _ ::  s', n'.+1 => drop n' s'
 | _, _ => s end.
Fixpoint take n s := match s, n with
 | x ::  s', n'.+1 => x ::  take n' s'
 | _, _ => [::]  end.
```

Intuitive...

A simple example

General library clustering:

SSReflect Base library, 12 standard files, 457 terms, 91 clusters (the number and size of clusters can be changed using PG options); 5-10 seconds.

Example

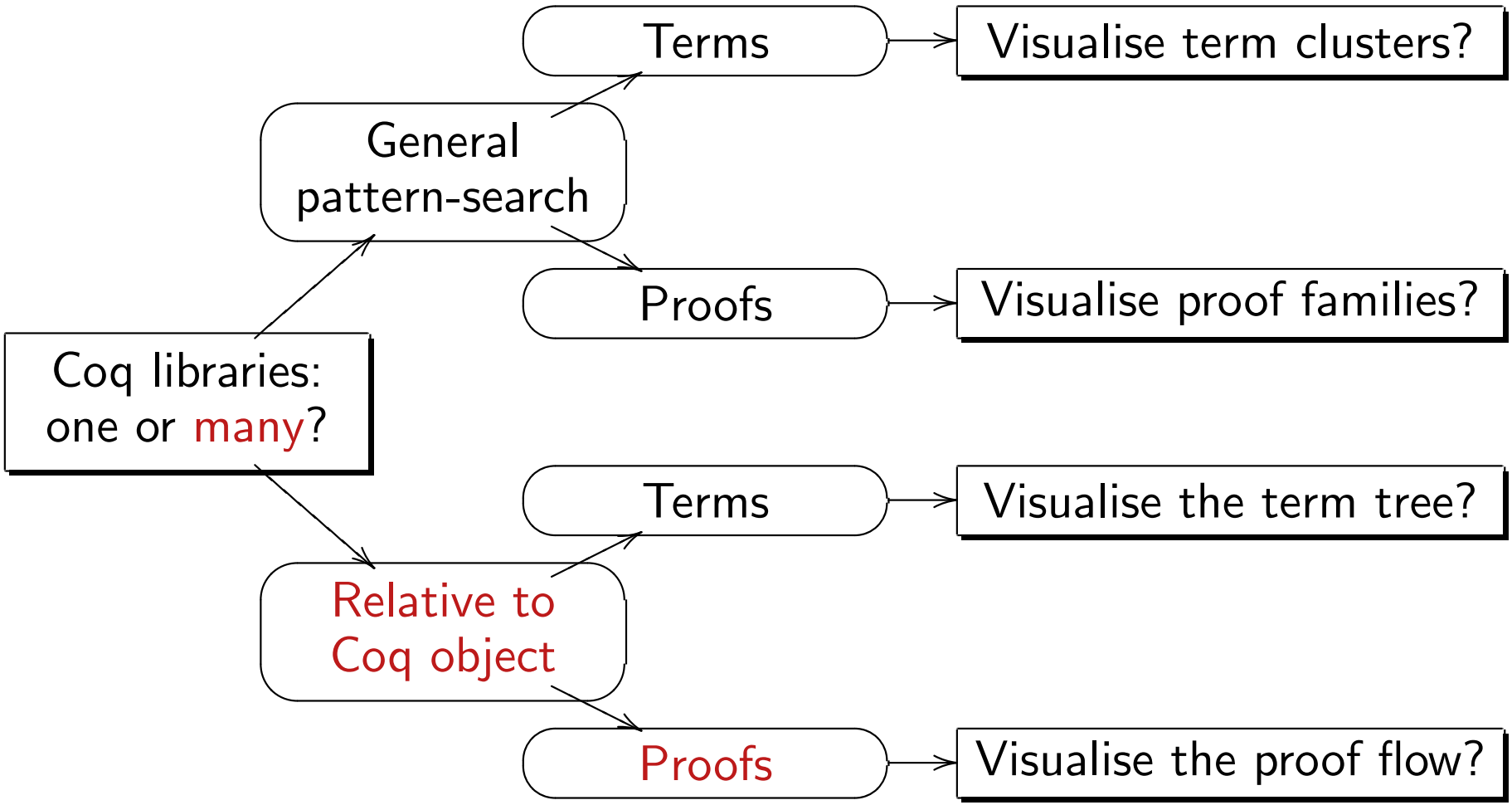
```
Definition flatten := foldr cat (Nil T).

Definition sumn := foldr addn 0.
```

Analyses deep into structures of subterms by recurrent clustering: `cat` and `addn` are defined on lists and natural numbers, but are in the same cluster recurrently. These 2 grouped together out of 15 other definitions using `foldr`.

- Goal-oriented clustering: do the same but show only what is related to certain Coq object: e.g. related to `flatten`. (See demo)

ML4PG


Terms
Visualise term clusters?
General pattern-search
Proofs
Visualise proof families?
Coq libraries: one or many?
Terms
Visualise the term tree?
Relative to Coq object
Proofs
Visualise the proof flow?

Proof-clustering

Similarly to term-clustering, the feature extraction:

- relies on recurrently computed features;
- considers a fragment of a proof-tree – a proof-patch – to find relative dependencies between goals, tactics and tactic arguments;
- considers relation of the tactic arguments to the (inductive) hypotheses or library lemmas.

As before,

- a 5-step proof patch is given by 85 features;
- each proof patch is a point in 85-dimensional space;
- the proof pattern is determined by looking at their correlation in several proofs.

A proof-feature algorithm by example

HoTT Path library

```
Lemma dpath_path_l A : Type x1 x2 y :  A
 (p :  x1 = x2) (q :  x1 = y) (r :  x2 = y) :
 q = p @ r <~> transport (fun x => x = y) p q = r.
Proof.
 destruct p; simpl.
 exact (equiv_concat_r (concat_1p r) q).
Qed.
Lemma transport_paths_lr A : Type x1 x2 :  A (p :  x1 = x2) (q :  x1 = x1)
 :  transport (fun x => x = x) p q = p^ @ q @ p.
Proof.
 destruct p; simpl.
 exact ((concat_1p q)^  @ (concat_p1 (1 @ q))^ ).
Qed.
```

A proof-feature algorithm by example

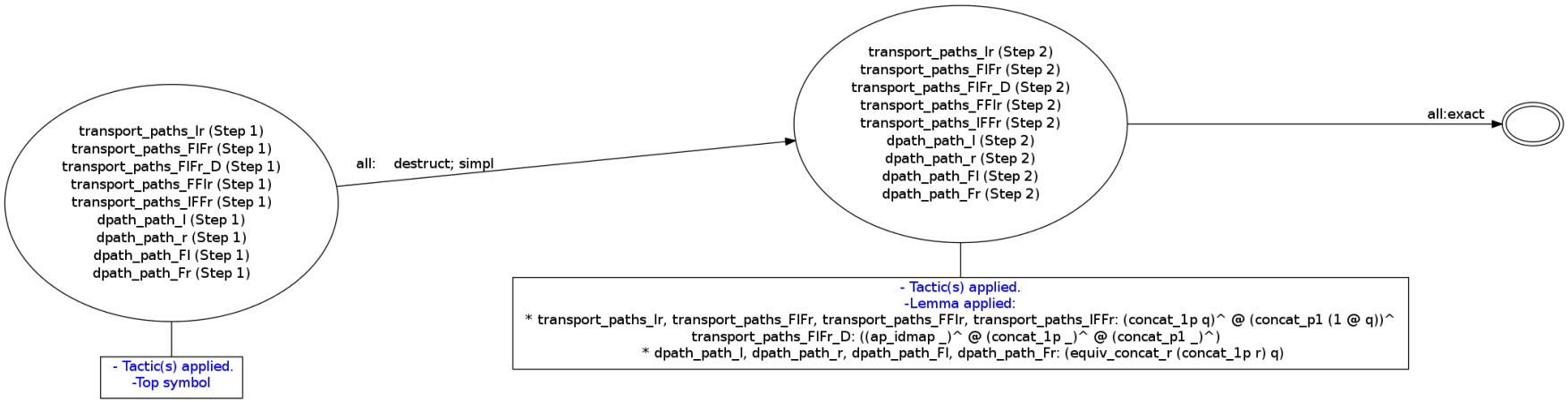
HoTT Path library

```
Lemma dpath_path_l A : Type x1 x2 y :  A
 (p :  x1 = x2) (q :  x1 = y) (r :  x2 = y) :
 q = p @ r <~> transport (fun x => x = y) p q = r.
Proof.
 destruct p; simpl.
 exact (equiv_concat_r (concat_1p r) q).
Qed.
```

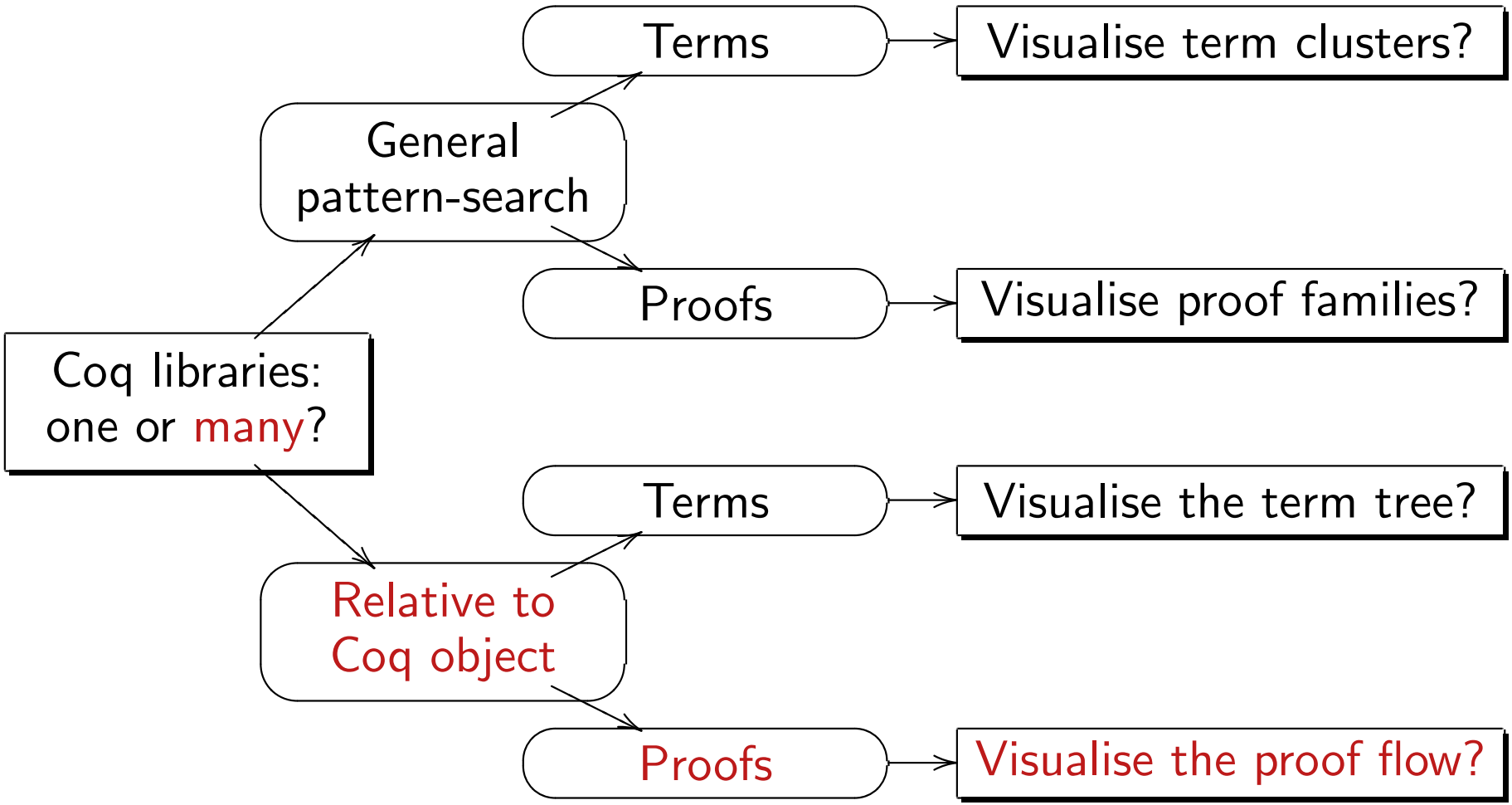
	tactics	*n*	*arg type*	*arg*	*symbols*	*goals*
g1	($[destruct]_{tac}$, $[simpl]_{tac}$, 0, 0)	2	($[\texttt{paths x1 x2}]_{type}$ 0, 0, 0)	($[\texttt{p}]_{term}$, 0, 0, 0)	($[\texttt{<\~{}>}]_{term}$, $[=]_{term}$, $[=]_{term}$)	1
g2	($[exact]_{tac}$, 0, 0, 0)	1	($[Prop]_{type}$, 0, 0, 0)	($[\texttt{(equiv\_concat\_r(concat\_1p r)q)}]_{term}$, 0, 0, 0)	($[\texttt{<\~{}>}]_{term}$, $[=]_{term}$, $[=]_{term}$)	0

Proof-patch analysis

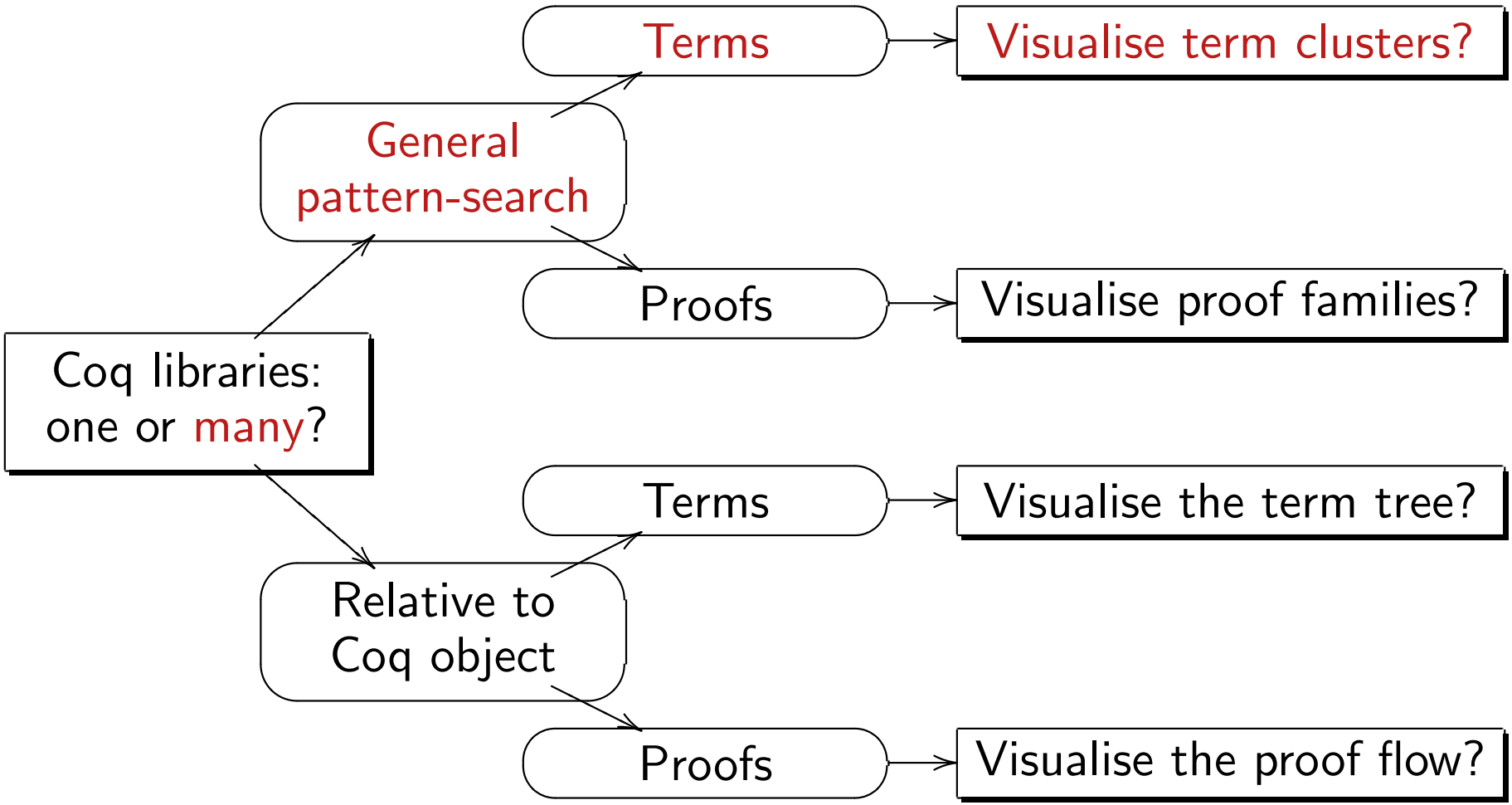
Knowing how to prove `Lemma dpath_path_l`, what else can I prove?



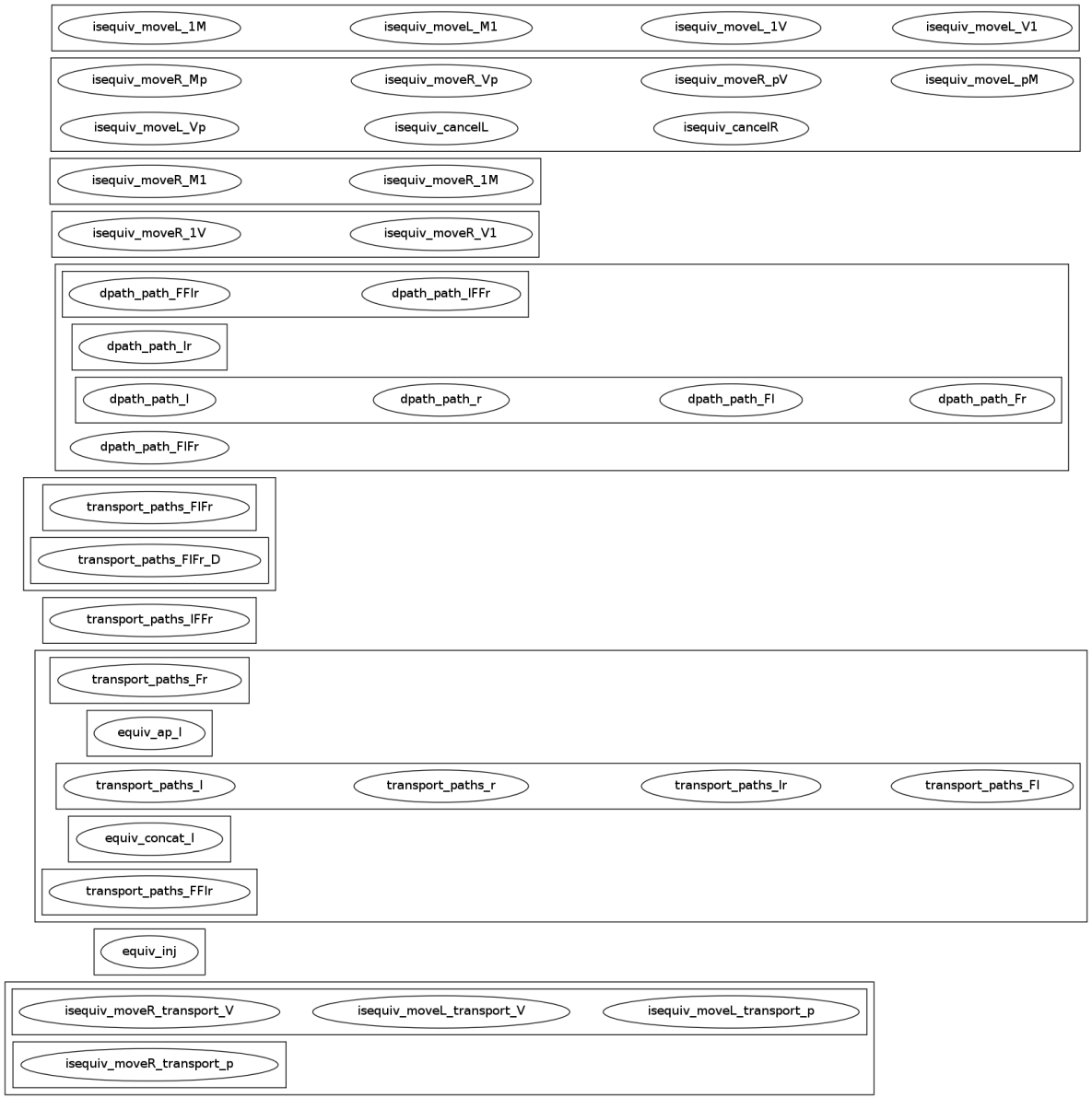
ML4PG: visualisation



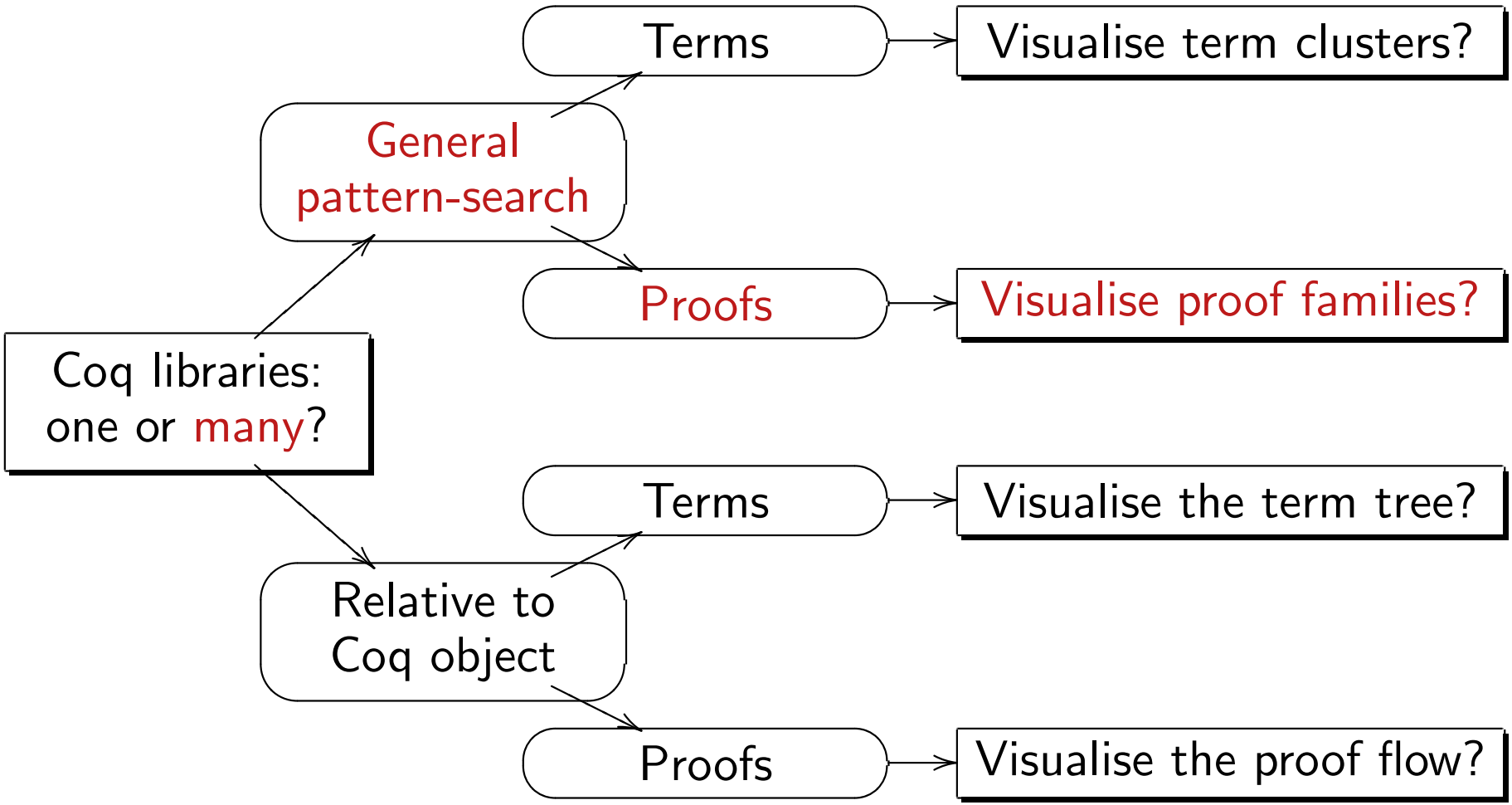
ML4PG: visualisation



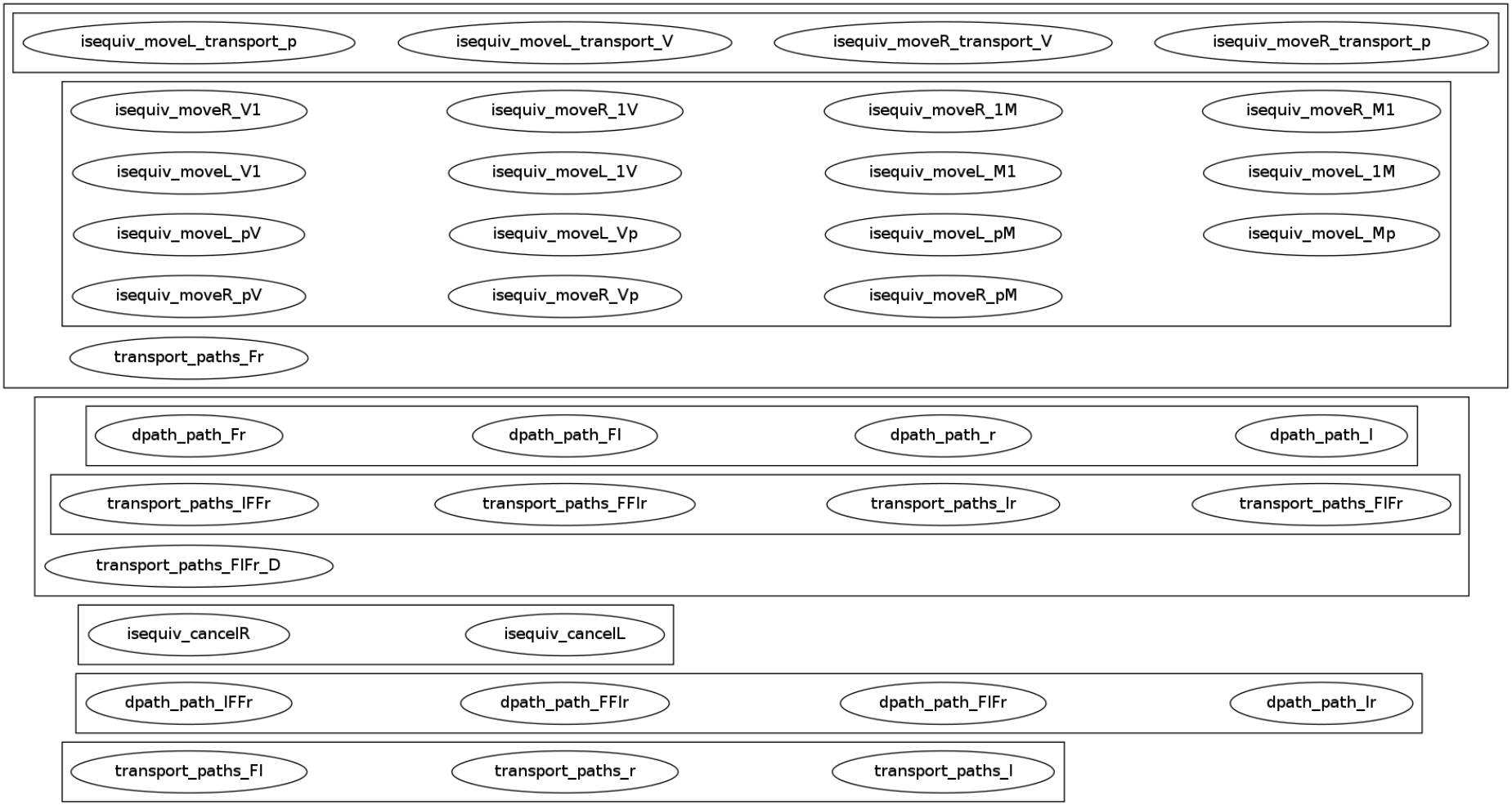
Visualisation: HoTT library Term Similarity Graph



ML4PG: visualisation



Visualisation: HoTT library Proof Similarity Graph



Outline

User Scenarios

- Detect proof-patterns prior to new library development

User Scenarios

- Detect proof-patterns prior to new library development
- Request a proof-hint during an on-going proof-development;

User Scenarios

- Detect proof-patterns prior to new library development
- Request a proof-hint during an on-going proof-development;
- Import a proof methodology from a different library;

User Scenarios

- Detect proof-patterns prior to new library development
- Request a proof-hint during an on-going proof-development;
- Import a proof methodology from a different library;
- Find how different proof strategies in two libraries about the same subject are;

User Scenarios

- Detect proof-patterns prior to new library development
- Request a proof-hint during an on-going proof-development;
- Import a proof methodology from a different library;
- Find how different proof strategies in two libraries about the same subject are;
- Share work-load in a team development.


J. Heras and K. Komendantskaya. ML4PG in Computer Algebra Verification. MKM/Calculemus/DML 2013: 354-358.


J. Heras and K. Komendantskaya. Recycling Proof-Patterns in Coq: Case Studies. MCS, 2014.

Benefits of this approach:

- ML4PG can be used on user's demand and in real-time;

Benefits of this approach:

- ML4PG can be used on user's demand and in real-time;
- does not assume any knowledge of machine-learning interfaces from the user;

Benefits of this approach:

- ML4PG can be used on user's demand and in real-time;
- 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 can be used on user's demand and in real-time;
- 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 can be used on user's demand and in real-time;
- 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.
- ML4PG is now a part of standard Proof General distribution.

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?			
Deterministic?			
Visual representation			
Covers proofs or terms?			
Takes into consideration dependencies?			
Finds structural similarities beyond concrete syntax?			
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?			
Visual representation			
Covers proofs or terms?			
Takes into consideration dependencies?			
Finds structural similarities beyond concrete syntax?			
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation			
Covers proofs or terms?			
Takes into consideration dependencies?			
Finds structural similarities beyond concrete syntax?			
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation	No	Yes	Yes
Covers proofs or terms?			
Takes into consideration dependencies?			
Finds structural similarities beyond concrete syntax?			
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation	No	Yes	Yes
Covers proofs or terms?	Terms/types	Both	Both
Takes into consideration dependencies?			
Finds structural similarities beyond concrete syntax?			
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation	No	Yes	Yes
Covers proofs or terms?	Terms/types	Both	Both
Takes into consideration dependencies?	No	Yes	Yes
Finds structural similarities beyond concrete syntax?			
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation	No	Yes	Yes
Covers proofs or terms?	Terms/types	Both	Both
Takes into consideration dependencies?	No	Yes	Yes
Finds structural similarities beyond concrete syntax?	No	No	Yes
Finds structurally similar patterns in proofs?			
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation	No	Yes	Yes
Covers proofs or terms?	Terms/types	Both	Both
Takes into consideration dependencies?	No	Yes	Yes
Finds structural similarities beyond concrete syntax?	No	No	Yes
Finds structurally similar patterns in proofs?	No	No	Yes
Good for proof automation?			

ML4PG in comparison

	Search	Dependency graphs	ML4PG
Method	Search	Parsing	Statistical Pattern-recognition
Goal-oriented?	Yes	Not necessarily	Not necessarily
Deterministic?	Yes	Yes	No
Visual representation	No	Yes	Yes
Covers proofs or terms?	Terms/types	Both	Both
Takes into consideration dependencies?	No	Yes	Yes
Finds structural similarities beyond concrete syntax?	No	No	Yes
Finds structurally similar patterns in proofs?	No	No	Yes
Good for proof automation?	No	No	?

Percentage of atomtaically re-proven theorems:

Library	Granularity 1	Granularity 3	Granularity 5
ssrnat (SSReflect)	48%	36%	28%
seq (SSReflect)	21%	20%	15%
ssrbool (SSReflect)	70%	77%	62%
fintype (SSReflect)	7%	7%	9%
JVM	56%	58%	65%
summations	0%	10%	12%
Paths (HoTT)	92%	91%	94%
Nash Equilibrium	40%	37%	36%

Outline

Statistical Machine Learning in Interactive Theorem Proving

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

University of Dundee

18 July 2014

Can we use ML4PG to automatically prove theorems?

Given the statement of a theorem T and the associated library L, we can use ML4PG to try to find a proof for T as follows:

1. Use ML4PG to obtain the cluster C from the library L that contains the theorem T.
2. Obtain the sequence of tactics $\{T_1^i, \ldots, T_{n_i}^i\}_i$ used to prove each lemma in C.
3. For each i, try to prove T using $T_1^i, \ldots, T_{n_i}^i$.
4. If no sequence of tactics prove T, then for each tactic use ML4PG to infer the argument for each tactic T_j^i:
   - If the argument of T_j^i is an internal hypothesis from the context of a proof, try all the internal hypothesis from the context of the current proof.
   - If the argument of T_j^i is an external lemma L, use ML4PG to compute all the lemmas in the same cluster as L and try all those lemmas.

*** This can be naturally extended to tactics with several arguments, just trying all the possible combinations.