ACL2(ml): Machine-Learning for ACL2

J. Heras and E. Komendantskaya

http://staff.computing.dundee.ac.uk/katya/acl2ml/

12 July 2014
ACL2'14

Outline

Outline

Some Challenges in ACL2

- Size of ACL2 library stands on the way of efficient knowledge reuse.
- Manual handling of proofs, strategies, libraries becomes difficult.
- Coordination of team-development can be hard.
- Comparison of proof similarities.
- Discovery of auxiliary lemmas can be difficult.

Some Challenges in ACL2

- Size of ACL2 library stands on the way of efficient knowledge reuse.
- Manual handling of proofs, strategies, libraries becomes difficult.
- Coordination of team-development can be hard.
- Comparison of proof similarities.
- Discovery of auxiliary lemmas can be difficult.

Could Machine-Learning help us to face some of these challenges?

- Statistical methods can discover patterns in proofs but are weak for conceptualisation.
- Symbolic methods (Proof planning, lemma discovery) can conceptualise but have limitations.

Some Challenges in ACL2

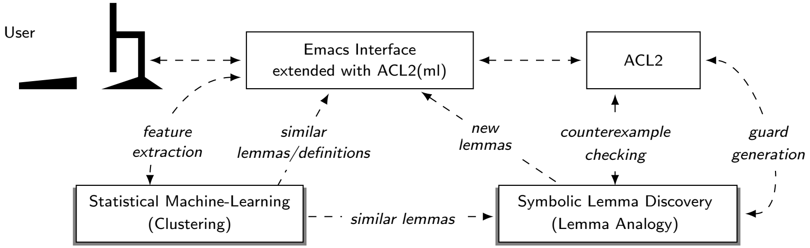
- Size of ACL2 library stands on the way of efficient knowledge reuse.
- Manual handling of proofs, strategies, libraries becomes difficult.
- Coordination of team-development can be hard.
- Comparison of proof similarities.
- Discovery of auxiliary lemmas can be difficult.

Could Machine-Learning help us to face some of these challenges?

- Statistical methods can discover patterns in proofs but are weak for conceptualisation.
- Symbolic methods (Proof planning, lemma discovery) can conceptualise but have limitations.
- Combination of statistical and symbolic methods:
  - Statistical methods can take advantage of symbolic methods to conceptualise results.
  - Symbolic tools can use statistical results for efficient lemma discovery.

Outline

ACL2(ml)



F.1. works on the background of Emacs extracting some low-level features from ACL2 definitions and theorems.

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

- **F.3.a** displays families of related proofs (or definitions) to the user.
- **F.3.b** uses the families of related proofs to discover new lemmas.

Outline

Extracting features from ACL2

- Feature extraction:

Extracting features from ACL2

- Feature extraction:
  - We extract features directly from term trees of ACL2 terms.

Definition (Term tree)

A variable or a constant is represented by a tree consisting of one single node, labelled by the variable or the constant itself. A function application $f(t_1, \dots, t_n)$ is represented by the tree with the root node labelled by f, and its immediate subtrees given by trees representing $t_1, \dots, t_n$.

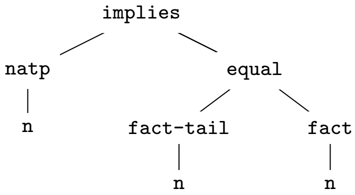
Extracting features from ACL2

- Feature extraction:
  - We extract features directly from term trees of ACL2 terms.

Definition (Term tree)

A variable or a constant is represented by a tree consisting of one single node, labelled by the variable or the constant itself. A function application $f(t_1, \ldots, t_n)$ is represented by the tree with the root node labelled by f, and its immediate subtrees given by trees representing $t_1, \ldots, t_n$.

```
(implies (natp n) (equal (fact-tail n) (fact n)))
```



ACL2(ml) term tree matrices

We have devised a compact feature extraction method.

ACL2(ml) term tree matrices

We have devised a compact feature extraction method.

Definition (Term tree depth level)

Given a term tree T, the *depth* of the node t in T, denoted by *depth(t)*, is defined as follows:
$-$ $depth(t) = 0$, if t is a root node;
$-$ $depth(t) = n + 1$, where n is the depth of the parent node of t.

ACL2(ml) term tree matrices

We have devised a compact feature extraction method.

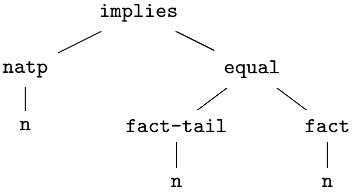
Definition (Term tree depth level)

Given a term tree T, the *depth* of the node t in T, denoted by *depth(t)*, is defined as follows:
– $depth(t) = 0$, if t is a root node;
– $depth(t) = n + 1$, where n is the depth of the parent node of t.

Definition (ACL2(ml) term tree matrices)

Given a term tree T for a term with signature Σ, and a function $[.] : \Sigma \to \mathbb{Q}$, the ACL2(ml) term tree matrix M_T is a 7×7 matrix that satisfies the following conditions:
– the $(0, j)$-th entry of M_T is a number $[t]$, such that t is a node in T, t is a variable and $depth(t) = j$.
– the (i, j)-th entry of M_T $(i \neq 0)$ is a number $[t]$, such that t is a node in T, t has arity $i + 1$ and $depth(t) = j$.

An example



	variables	arity 0	arity 1	arity 2
$td0$	0	0	0	[implies]
$td1$	0	0	[natp]	[equal]
$td2$	[n]	0	[fact-tail]::[fact]	0
$td3$	[n]::[n]	0	0	0

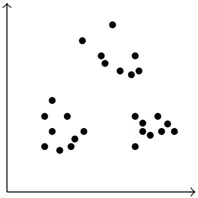
Clustering in ACL2(ml)

We have integrated Emacs with a variety of clustering algorithms:

Clustering in ACL2(ml)

We have integrated Emacs with a variety of clustering algorithms:

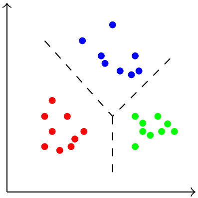
- Unsupervised machine learning technique:



Clustering in ACL2(ml)

We have integrated Emacs with a variety of clustering algorithms:

- Unsupervised machine learning technique:

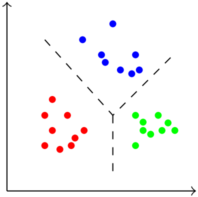


- Engines: Matlab, Weka, Octave, R, . . .

Clustering in ACL2(ml)

We have integrated Emacs with a variety of clustering algorithms:

- Unsupervised machine learning technique:

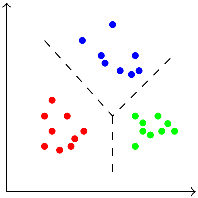


- Engines: Matlab, Weka, Octave, R, ...

Clustering in ACL2(ml)

We have integrated Emacs with a variety of clustering algorithms:

- Unsupervised machine learning technique:

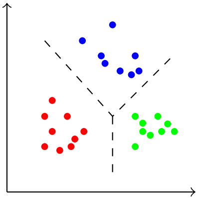


- Engines: Matlab, Weka, Octave, R, ...
- Algorithms: K-means, simple Expectation Maximisation, ...

Clustering in ACL2(ml)

We have integrated Emacs with a variety of clustering algorithms:

- Unsupervised machine learning technique:



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

Recurrent clustering

Three kinds of function symbols:

- Built-in functions: predefined value.
- Variables: based on the De Bruijn index.
- Functions defined on terms of other functions: recurrent clustering process.
    - Recursive and mutually-recursive function occurrences have a fixed value.

Demo

- Finding similar theorems across libraries.
- Obtaining more precise clusters.
- Finding similar definitions across libraries.

Outline

Lemma analogy in ACL2(ml)*

Can we use the output of the statistical side of ACL2(ml) to generate useful lemmas?

*Joint work with E. Maclean and M. Johansson

Lemma analogy in ACL2(ml)*

Can we use the output of the statistical side of ACL2(ml) to generate useful lemmas?
Terminology:

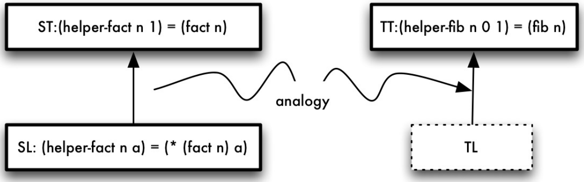
- Target Theorem (TT): the theorem that we want to prove.
- Source Theorem (ST): theorem suggested as similar to TT.
- Source Lemma (SL): a user-supplied lemma to prove ST.

*Joint work with E. Maclean and M. Johansson

Lemma analogy in ACL2(ml)*

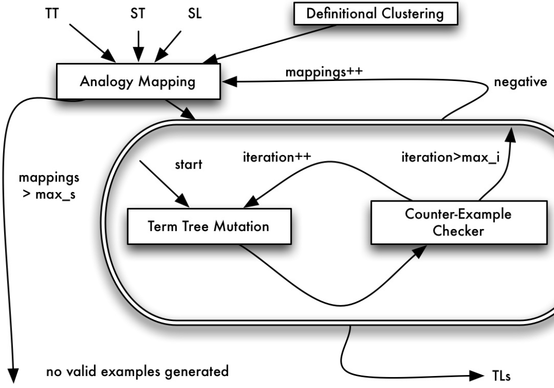
Can we use the output of the statistical side of ACL2(ml) to generate useful lemmas?
Terminology:

- Target Theorem (TT): the theorem that we want to prove.
- Source Theorem (ST): theorem suggested as similar to TT.
- Source Lemma (SL): a user-supplied lemma to prove ST.



* Joint work with E. Maclean and M. Johansson

Overview of the process



Using guards to generate preconditions

- Using the lemma analogy tool, ACL2(ml) generates the following suggestion:

```
(equal (helper_fib n j k)
       (+ (* (theta_fib (- n 1)) j) (* (theta_fib n) k)))
```

Using guards to generate preconditions

- Using the lemma analogy tool, ACL2(ml) generates the following suggestion:

```
(equal (helper_fib n j k)
       (+ (* (theta_fib (- n 1)) j) (* (theta_fib n) k))))
```

- This result cannot be directly proven in ACL2, we need some preconditions.

Using guards to generate preconditions

- Using the lemma analogy tool, ACL2(ml) generates the following suggestion:

```
(equal (helper_fib n j k)
       (+ (* (theta_fib (- n 1)) j) (* (theta_fib n) k))))
```

- This result cannot be directly proven in ACL2, we need some preconditions.
- In ACL2, we can restrict a function to a particular domain using the guard mechanism.

Using guards to generate preconditions

- Using the lemma analogy tool, ACL2(ml) generates the following suggestion:

```
(equal (helper_fib n j k)
       (+ (* (theta_fib (- n 1)) j) (* (theta_fib n) k))))
```

- This result cannot be directly proven in ACL2, we need some preconditions.
- In ACL2, we can restrict a function to a particular domain using the guard mechanism.
- Guards are optional and several functions do not include them.
- ACL2 recommendation for novices: "novices are often best served by avoiding guards".

Using guards to generate preconditions

- Using the lemma analogy tool, ACL2(ml) generates the following suggestion:

```
(equal (helper_fib n j k)
       (+ (* (theta_fib (- n 1)) j) (* (theta_fib n) k))))
```

- This result cannot be directly proven in ACL2, we need some preconditions.
- In ACL2, we can restrict a function to a particular domain using the guard mechanism.
- Guards are optional and several functions do not include them.
- ACL2 recommendation for novices: "novices are often best served by avoiding guards".
- Solution: compute recursively the guards of a function f.

Using guards to generate preconditions

```
(defun helper_fib (n j k)
     (if (zp n) j (if (equal n 1) k (helper_fib (- n 1) k (+ j k)))))

  * zp ->  (natp x)
  * equal -> t
  * + -> (and (acl2-numberp x) (acl2-numberp y))
  * - -> (and (acl2-numberp x) (acl2-numberp y))
```

Using guards to generate preconditions

```
(defun helper_fib (n j k)
     (if (zp n) j (if (equal n 1) k (helper_fib (- n 1) k (+ j k)))))

  * zp ->  (natp x)
  * equal -> t
  * + -> (and (acl2-numberp x) (acl2-numberp y))
  * - -> (and (acl2-numberp x) (acl2-numberp y))
```

Guards generated for `helper_fib` $\rightarrow$

```
(and (natp n) t (and (acl2-numberp n) (acl2-numberp 1))
     (and (acl2-numberp j) (acl2-numberp k)))
```

$\xrightarrow{simpl}$ `(and (integerp n) (not (< n 0)) (acl2-numberp j) (acl2-numberp k))`

Using guards to generate preconditions

```
(defun helper_fib (n j k)
     (if (zp n) j (if (equal n 1) k (helper_fib (- n 1) k (+ j k)))))
```

```
  * zp ->  (natp x)
  * equal -> t
  * + -> (and (acl2-numberp x) (acl2-numberp y))
  * - -> (and (acl2-numberp x) (acl2-numberp y))
```

Guards generated for `helper_fib` $\rightarrow$

```
(and (natp n) t (and (acl2-numberp n) (acl2-numberp 1))
     (and (acl2-numberp j) (acl2-numberp k)))
```

$\xrightarrow{\textit{simpl}}$ `(and (integerp n) (not (< n 0)) (acl2-numberp j) (acl2-numberp k))`

```
(defthm helper_fib_theta_fib
   (equal (helper_fib n j k)
          (+ (* (theta_fib (- n 1)) j) (* (theta_fib n) k))))
```

Guards:

```
(and (integerp n) (not (< n 0)) (acl2-numberp j) (acl2-numberp k)
     (not (< (+ -1 n) 0)))
```

Demo

- Lemma discovery.
- Guard generation.

Outline

Benefits of ACL2(ml)

- ACL2(ml) statistical and symbolic tools can be switched on/off on user's demand;

Benefits of ACL2(ml)

- ACL2(ml) statistical and symbolic tools can be switched on/off on user's demand;
- ACL2(ml) does not assume any knowledge of machine-learning from the user;

Benefits of ACL2(ml)

- ACL2(ml) statistical and symbolic tools can be switched on/off on user's demand;
- ACL2(ml) does not assume any knowledge of machine-learning from the user;
- modular: allows the user to make choices regarding approach to levels of proofs and particular statistical algorithms;

Benefits of ACL2(ml)

- ACL2(ml) statistical and symbolic tools can be switched on/off on user's demand;
- ACL2(ml) does not assume any knowledge of machine-learning 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 ACL2(ml)

- ACL2(ml) statistical and symbolic tools can be switched on/off on user's demand;
- ACL2(ml) does not assume any knowledge of machine-learning 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.

Conclusions

- ACL2(ml) combines statistical machine learning (detection of patterns) with symbolic techniques (generation of lemmas).
- ACL2(ml) is different to other tools:
  - its methods of generating the proof-hints interactively and in real-time;
  - its flexible environment for integration of statistical and symbolic techniques.

Further work

- **Reimplement ACL2(ml) as ACL2 book.** All ACL2(ml) modules are currently implemented in Emacs Lisp.
- **Use of information generated by failed proof-attempts.**
- **Different patterns.** Statistical ACL2(ml) groups in the same clusters theorems whose lemmas cannot be mutated to generate any useful lemma.
- **Smaller lemmas.** The lemma analogy tool currently only adds term structure; therefore, it cannot generate smaller lemmas.
- **Conditional lemmas.** Discovering appropriate conditions for generated lemmas is a difficult problem for theory exploration systems.
- **New definitions.** Another big challenge in lemma discovery is the invention of new concepts.

ACL2(ml): Machine-Learning for ACL2

J. Heras and E. Komendantskaya

http://staff.computing.dundee.ac.uk/katya/acl2ml/

12 July 2014
ACL2'14

How is the function [.] defined?

How is the function [.] defined?

Definition (Function [.])

Given the nth term definition of the library (call the term t), a function [.] is inductively defined for every symbol s in t as follows:
$-$ $[s] = i$, if s is the ith distinct variable in t (formulas are implicitly universally quantified);
$-$ $[s] = -[m]$, if t is a recursive definition defining the function s with measure function m;
$-$ $[s] = k$, if s is a function imported from CLISP; and $[s] = k$ in the figure below;
$-$ $[s] = 5 + 2 \times j + p$, where C_j is a cluster obtained as a result of definition clustering with granularity 3 for library definitions 1 to $n-1$, $s \in C_j$ and p is the proximity value of s in C_j.

$*$ Type recognisers (r = {symbolp, characterp, stringp, consp, acl2-numberp, integerp, rationalp, complex-rationalp}): $[r_i] = 1 + \sum_{j=1}^{i} \frac{1}{10 \times 2^{j-1}}$ (where r_i is the ith element of r).

$*$ Constructors (c = {cons, complex}): $[c_i] = 2 + \sum_{j=1}^{i} \frac{1}{10 \times 2^{j-1}}$.

$*$ Accessors (a^1 = {car, cdr}, a^2 = {denominator, numerator}, a^3 = {realpart, imagpart}): $[a_i^j] = 3 + \frac{1}{10 \times j} + \frac{i-1}{100}$.

$*$ Operations on numbers (o = {unary-/, unary–, binary-+, binary-*}): $[o_i] = 4 + \sum_{j=1}^{i} \frac{1}{10 \times 2^{j-1}}$.

$*$ Integers and rational numbers: $[0] = 4.3$, $[n] = 4.3 + \frac{|n|}{10}$ (with $n \neq 0$ and $|n| < 1$) and $[n] = 4.3 + \frac{1}{100 * |n|}$ (with $n \neq 0$ and $|n| > 1$).

Analogy mapping

Definition (Analogy Mapping $\mathcal{A}$)

For all symbols $s_1, \ldots, s_n$ occurring in the current ST, the set of admissible analogy mappings is the set of all mappings $\mathcal{A}$ such that
- $\mathcal{A}(s_i) = s_i$ for all shared background symbols; otherwise:
- $\mathcal{A}(s_i) = s_j$ for all combinations of $i, j \in 1 \ldots n$, such that s_i and s_j belong to the same cluster in the last iteration of definition clustering.

Analogy mapping

Definition (Analogy Mapping $\mathcal{A}$)

For all symbols $s_1, \ldots, s_n$ occurring in the current ST, the set of admissible analogy mappings is the set of all mappings $\mathcal{A}$ such that
- $\mathcal{A}(s_i) = s_i$ for all shared background symbols; otherwise:
- $\mathcal{A}(s_i) = s_j$ for all combinations of $i, j \in 1 \ldots n$, such that s_i and s_j belong to the same cluster in the last iteration of definition clustering.

Example

For our running example, the shared background theory includes symbols {+, *,-, 1, 0}. We thus get a mapping:
$\mathcal{A} = \{$`fact` $\mapsto$ `fib`, `helper-fact` $\mapsto$ `helper-fib`, `+` $\mapsto$ `+`, `1` $\mapsto$ `1`,...$\}$

Term tree mutation

Term tree mutation consists of three iterations:

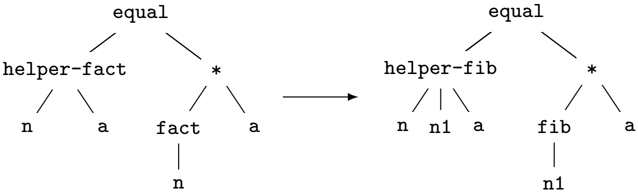
- Tree reconstruction.
- Node expansion.
- Term tree expansion.

Tree reconstruction

Tree Reconstruction phase replaces symbols in the SL with their analogical counterparts.

Tree reconstruction

Tree Reconstruction phase replaces symbols in the SL with their analogical counterparts.

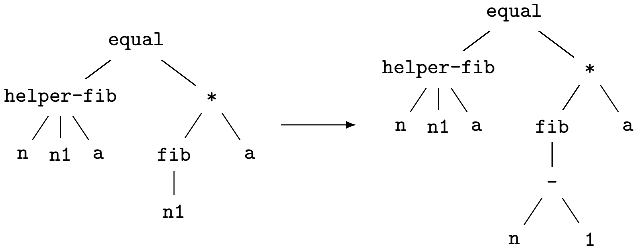


Node expansion

Node expansion phase mutates the term, by synthesising small terms (max depth 2) in place of variables.

Node expansion

Node expansion phase mutates the term, by synthesising small terms (max depth 2) in place of variables.



Term Tree Expansion

Term Tree Expansion phase is similar to Node expansion phase, but adding new term structure on the top-level of the term.

Term Tree Expansion

Term Tree Expansion phase is similar to Node expansion phase, but adding new term structure on the top-level of the term.