Automated theorem provers for ACL2

Sebastiaan J.C. Joosten
Cezary Kaliszyk
Josef Urban

Open University of the Netherlands, Heerlen
Radboud University Nijmegen
University of Innsbruck

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Automated Theorem Proving for ACL2?

- ACL2 is a strong automated theorem prover by itself
- ACL2 has a fast executable counterpart, useful for testing
- Certain theorems are proved with dedicated meta-rules
- ATPs cannot perform induction on ACL2's structures
- An ATP's performance degrades heavily as the set of known theorems increases

- Looks like ATPs probably won't help ACL2 users…
  - So why do it?

ATP: Automated Theorem Prover

Automated Theorem Proving for ACL2?

- ACL2 is a strong automated theorem prover by itself
- ACL2 has a fast executable counterpart, useful for testing
- Certain theorems are proved with dedicated meta-rules
- ~~ATPs cannot perform induction on ACL2's structures~~
- ~~An ATP's performance degrades heavily as the set of known theorems increases~~
- ATPs keep getting better. They are improving fast!

ATP: Automated Theorem Prover

Automated Theorem Proving for ACL2

- Test which ATP techniques will benefit ACL2
- Help the ATP community get new problem-sets
- Push the ATP community to develop techniques that benefit ACL2
- Get a feeling for how ACL2 compares to ATP

Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

Contributions

TPTP: Thousand problems for Theorem Proving

Previously developed ACL2 theorems

Translate to TPTP

New TPTP problems

New ACL2 theorem solving techniques

Select useful ones

Previously developed TPTP solving techniques

How do we re-prove theorems?

- LD the ACL2 book from which we get the theorems
- Extract all definitions and theorems from the ACL2 world
- Extract the proof tree from the ACL2 world
- Convert the ACL2 theorems into a TPTP format
- Preprocess the TPTP files
- Re-attempt each proof using the proof dependencies for that theorem
- Re-attempt each proof without using the proof tree for that theorem

Extracting from the ACL2 world

```
; load the original book
(ld '((ld “'FILE.acl2"  :ld-error-action :error))
    :ld-error-action :continue) ; catch-all to continue
(ld '((ld "cert.acl2"  :ld-error-action :error))
    :ld-error-action :continue) ; catch-all to continue
(ld "'FILE.lisp" :ld-error-action :error) ; no catch-all
(set-raw-mode-on!)
(defconst *w* (w state))
…
```

(some auxiliary functions)

```
…
(defconst *gp* (acl2::fgetprop 'ACL2::PROOF-SUPPORTERS-ALIST
'ACL2::global-value NIL *w*))
(appendprops (remove-duplicates (acl2::append-lst *gp*))
             (cons (cons ':PROOF-TREE (reverse *gp*)) nil)
             *w*)
```

Pros and Cons for this extraction method

- No macro's and other scary stuff
  - Pro: improved soundness
  - Con: some macro's generate definitions and theorems that are of no interest to the user
- All theorems are replayed
  - Pro: we are very certain that all theorems hold
  - Con: takes a long time to get all theorems in this way (800 min)
- Local encapsulated theorems/definitions are not exported
  - Pro: no cheating
  - Con: proofs using functional-instance-of cannot be reproduced using the original proof tree
  - Con: almost all encapsulated non-local theorems become disprovable

Translating ACL2 to TPTP

- TPTP is a lot like Prolog in the way that ACL2 is like LISP
- Typical ACL2 definition (written as a theorem):

```
(EQUAL (E0-ORDINALP X)
       (IF (CONSP X)
         (IF (E0-ORDINALP (CAR X))
           (IF (EQUAL (CAR X) '0)
             'NIL
             (IF (E0-ORDINALP (CDR X))
               (IF (CONSP (CDR X))
                 (IF (E0-ORD-< (CAR X)
                               (CAR (CDR X)))
                   'NIL 'T) 'T) 'NIL)) 'NIL)
         (IF (INTEGERP X)
           (IF (< X '0) 'NIL 'T) 'NIL)))
```

Translating ACL2 to TPTP

- TPTP version:

```
fof(e0_ordinalp,axiom,
   (acleq(e0_ordinalp(X),
          if(consp(X),
             if(e0_ordinalp(car(X)),
                if(acleq(car(X),0),nil,
                   if(e0_ordinalp(cdr(X)),
                      if(consp(cdr(X)),
                         if('e0_ord_<'(car(X),car(cdr(X))),nil,t),t),nil)),nil),
          if(integerp(X),if('<'(X,0),nil,t),nil)))
!= nil)).
```

Translating ACL2 to TPTP

- Basic axioms of ACL2:

```
fof(spcax1,axiom, t != nil).
fof(spcax2,axiom, ! [X,Y]: ((X = Y) <=> acleq(X,Y) = t)).
fof(spcax3,axiom, ! [X,Y]: ((X != Y) <=> acleq(X,Y) = nil)).
fof(spcax4,axiom, ! [B,C]: (if(nil,B,C) = C)).
fof(spcax5,axiom, ! [A,B,C]: ((A != nil) => if(A,B,C) = B)).
fof(spcax6,axiom, ! [A]: (not(A) = if(A, nil, t))).
fof(spcax7,axiom, ! [P,Q]: (implies(P,Q) = if(P,if(Q,t,nil),t))).
fof(spcax8,axiom, ! [P,Q]: (iff(P,Q) =
                                    and(implies(P,Q),implies(Q,P)))).
fof(and,axiom, ! [A,B]: (and(A,B) = if(A,B,nil))).
```

Trouble in the ACL2 to TPTP translation

- The ‘function’ quote.
- Most constants are distinct (eg 1≠2, NIL≠cons(X,Y) etc). This was not implemented (except for T ≠ NIL).
- Some functions have no definition in the ACL2 logical world. Fortunately, the constructor/destructor theorems usually are.
- Defchoose and local encapsulated information is ignored.

Pre-processing TPTP files

- TPTP originally:

```
fof(e0_ordinalp,axiom,
   (acleq(e0_ordinalp(X),
          if(consp(X),
             if(e0_ordinalp(car(X)),
               if(acleq(car(X),0),nil,
                  if(e0_ordinalp(cdr(X)),
                     if(consp(cdr(X)),
                        if('e0_ord_<'(car(X),car(cdr(X))),nil,t),t),nil)),nil),
          if(integerp(X),if('<'(X,0),nil,t),nil)))
!= nil)).
```

Pre-processing TPTP files

- TPTP new version:

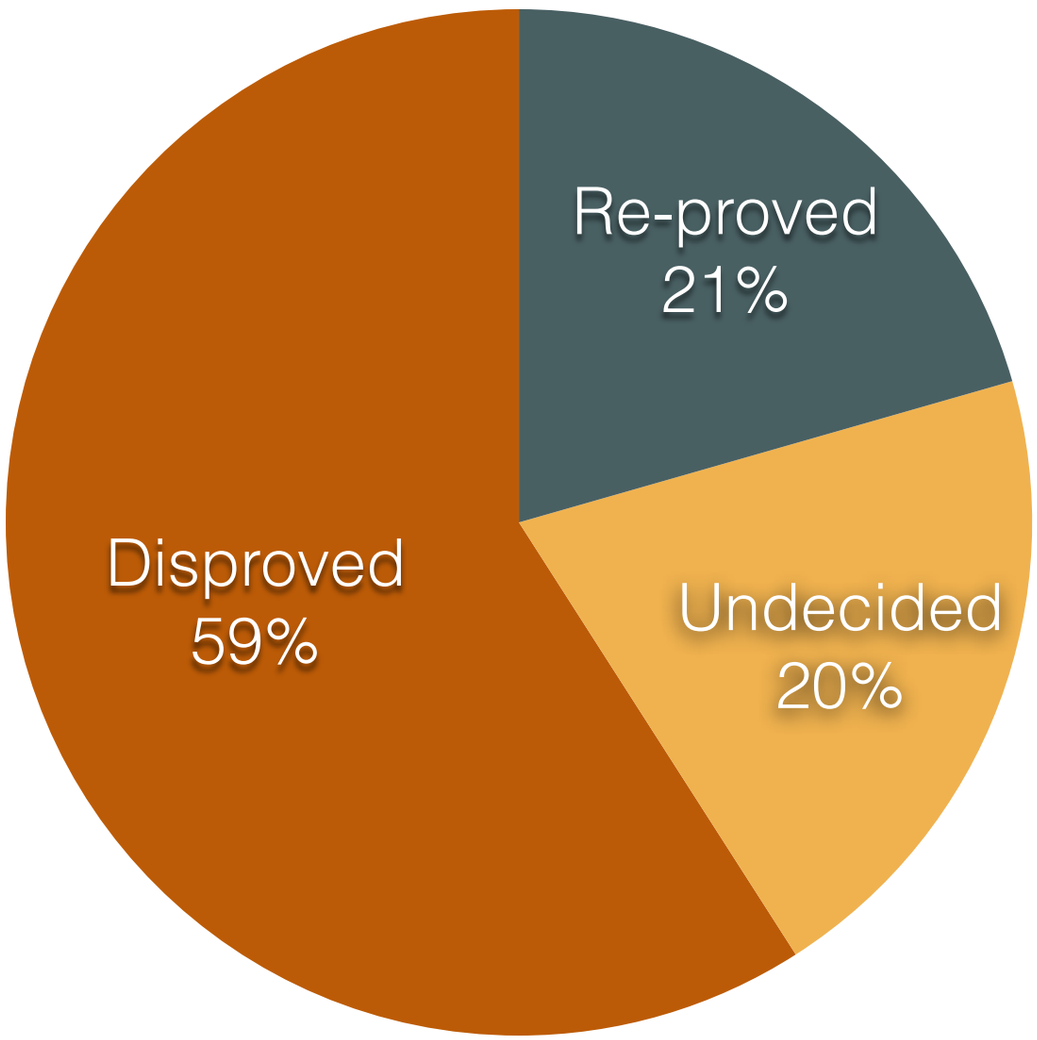
```
fof(e0_ordinalp,axiom, ! [X]:
   (acleq(e0_ordinalp(X),
          if(consp(X),
             if(e0_ordinalp(car(X)),
               if(acleq(car(X),0),nil,
                  if(e0_ordinalp(cdr(X)),
                    if(consp(cdr(X)),
                       if('e0_ord_<'(car(X),car(cdr(X))),nil,t),t),nil)),nil),
          if(integerp(X),if('<'(X,0),nil,t),nil)))
!= nil)).
```

Re-attempt with original proof dependencies

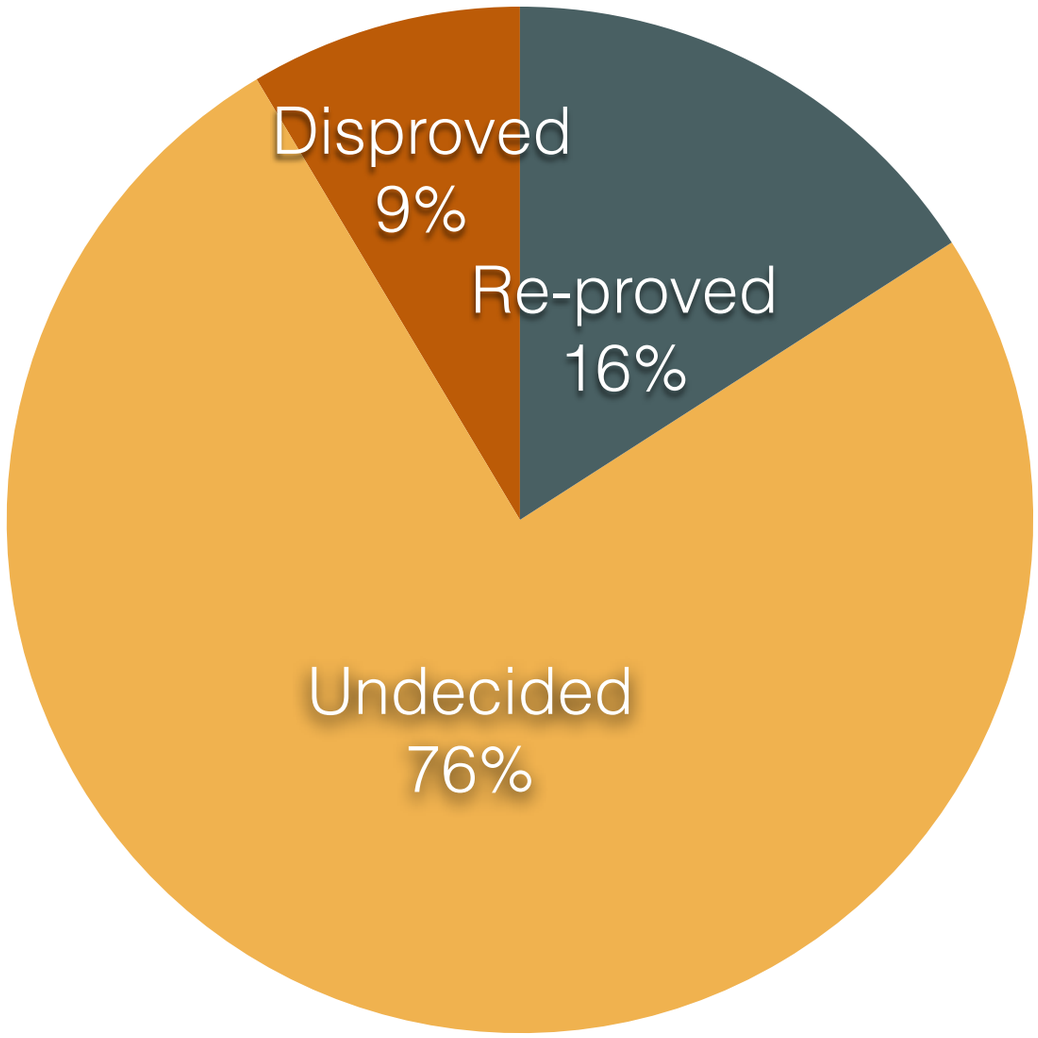
- Two problems:
  - Built-in data types (numbers) and functions on them.
  - ACL2 is a higher-order theorem prover!
    - :functional-instance
    - defchoose
      - this induction scheme is justified by the same argument used to admit …
    - meta rules
    - … ?
- Some theorems have finite counter-models
- Some theorems are not provable in first order logic

Re-attempt with original proof dependencies

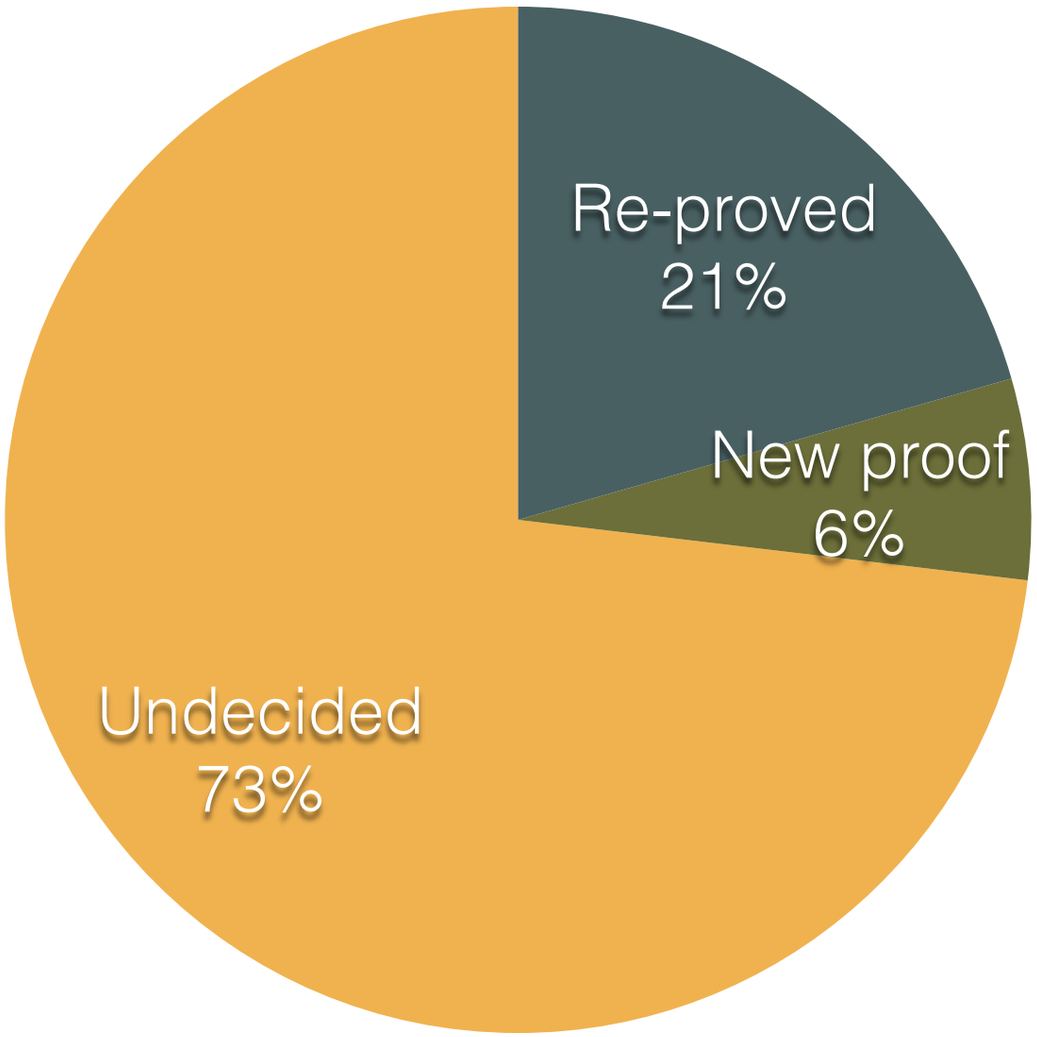
Use the same lemmas as the original proof



Using 100 computer-picked lemmas



Combining with computer-picked lemmas



How well are the lemma's picked? (on average)

	Lemmas in ACL2's proof tree	Lemmas not in ACL2's proof tree
Lemmas selected by computer	8	92
Lemmas not selected	1	

Conclusions

- ACL2 is a powerful theorem prover!
  - Particularly on theorems it has already proved :-)
- Exporting of the proof tree should be improved
  - It would help if ACL2 would improve its accounting
- Our lemma picking algorithm works great on ACL2
  - This is largely automated in ACL2, so perhaps our machine learning picks up on that