The ACL2 Proof Builder

https://acl2.org/manual See topic: proof-builder

Warren A. Hunt, Jr.
hunt@cs.utexas.edu

Computer Science Department
University of Texas
2317 Speedway, M/S D9500
Austin, TX 78712-0233

February, 2026

Example Usage

Consider functions that counts CONS pairs and ATOMs (tips).

```
(defun cc (x)
  ;; If atom recognized
  (if (atom x)
      0  ;; No CONS pairs
    ;; otherwise, add 1 to the CONS pairs in the subtrees
    (+ 1
       (cc (car x))
       (cc (cdr x)))))
```

```
(defun atoms (x)
  ;; If atom recognized
  (if (atom x)
      1  ;; One atom
    ;; otherwise, sum tips in the left and right subtrees
    (+ (atoms (car x))
       (atoms (cdr x)))))
```

Now, that we have defined these two concepts, can you think of a relationship between these functions?

There is one more ATOM than CONS Pairs

Let's conjecture that there is one more tip (atom) than CONS pairs. Likely, you were exposed to a proof of this in your discrete math class.

We will state this relationship as follows:

```
(defthm atoms-and-cc-related
  (equal (atoms x)
         (1+ (cc x))))
```

This conjecture states that, for every ACL2 object, there one more tip than there are tips (atoms).

Can we prove this?

For these first few problems, it's probably a good idea to prove such a conjecture by hand – and then attempt to orchestrate the Proof Builder.

Said another way, it's easier to overcome one new thing at a time.

Starting the ACL2 Proof Builder

The ACL2 Proof Builder is part of the ACL2 theorem-proving system.

It allows a user to use ACL2 in a rather manual way.

A user provides a conjecture with the `verify` command:

```
(verify (equal (atom x) (1+ (cc x))))
```

This problem has a straight-forward induction scheme. There is only one variable, `x`, so we are limited in our selection.

Our first instruction will be the `:induct` or the `(:induct)` command.

Please read the output carefully. After a while, you will be able to see if a proof is likely to succeed.

You will notice that ACL2 has broken the proof into two parts: a base case and an induction case. Why? Because that's what you told ACL2 to do!

`See topic: proof-builder-commands-short-list`

Initially, we confine ourselves to the short list of Proof-Builder commands

The ACL2 Proof Builder State

The Proof Builder keeps a stack of goals. Initially, a user supplies a single goal.

The goals are cooperatively managed by the Proof Builder and the user.

The `:induct` command will instantiate the **Rule of Induction**, but with our simple, list-focused case, it will conform to the **Rule of Structural Induction**.

Let's have a look at the two cases.

Just like in class, we may choose to consider the base case first. One can reorder the goals on the stack of goals.

Our mission is to eliminate all goals; once done, our proof is finished.

We will now consider how to complete this proof. [Interactive Demo]

ACL2 PB: Goal Stack and its Management

The ACL2 Proof-Builder keeps the remaining goals on a stack.

The top (of the stack) goal is the active goal; users can inspect the goals and alter the order of the goals.

Goals are presented as a list of hypotheses with a conclusion.

```
[ACL2-pc::cg]
    (macro) change to another goal.

[ACL2-pc::goals]
    (macro) list the names of goals on the stack

[ACL2-pc::p]
    (macro) prettyprint the current term in the usual user-level
    (untranslated) syntax

[ACL2-pc::th]
    (macro) print the top-level hypotheses and the current subterm
```

We can alter the order we prosecute the goals.

ACL2 PB: Movement Within a Goal

A user may move the Proof Builder focus within a goal.

```
[ACL2-pc::bk]
    (atomic macro) move backward one argument in the enclosing term

[ACL2-pc::dv]
    (atomic macro) move to the indicated subterm

[ACL2-pc::nx]
    (atomic macro) move forward one argument in the enclosing term

[ACL2-pc::p-top]
    (macro) prettyprint the conclusion, highlighting the current term

[ACL2-pc::top]
    (atomic macro) move to the top of the goal

[ACL2-pc::up]
    (primitive) move to the parent (or some ancestor) of the current
    subterm
```

To see the goal(s), use the **Goal Stack and its Management** commands.

ACL2 PB: Proof Commands

For the next few weeks, commands marked with "-" may not be used. And commands marked with "*" may only be used in restricted ways.

```
* [ACL2-pc::=]
      (atomic macro) attempt an equality (or equivalence) substitution

- [ACL2-pc::bash]
      (atomic macro) call the ACL2 theorem prover's simplifier

* [ACL2-pc::claim]
      (atomic macro) add a new hypothesis

  [ACL2-pc::contrapose]
      (primitive) switch a hypothesis with the conclusion, negating both

  [ACL2-pc::demote]
      (primitive) move top-level hypotheses to the conclusion

  [ACL2-pc::drop]
      (primitive) drop top-level hypotheses

  [ACL2-pc::expand]
      (primitive) expand the current function call without simplification

  [ACL2-pc::induct]
      (atomic macro) generate subgoals using induction
```

ACL2 PB: Proof Commands, continued

```
- [ACL2-pc::prove]
      (primitive) call the ACL2 theorem prover to prove the current goal

  [ACL2-pc::r]
      (macro) same as rewrite

  [ACL2-pc::s]
      (primitive) simplify the current subterm

  [ACL2-pc::s-prop]
      (atomic macro) simplify propositionally

  [ACL2-pc::show-rewrites]
      (macro) display the applicable [rewrite] rules

  [ACL2-pc::sr]
      (macro) same as SHOW-REWRITES

  [ACL2-pc::use]
      (atomic macro) use a lemma instance

  [ACL2-pc::x]
      (atomic macro) expand and (maybe) simplify function call at the
      current subterm

  [ACL2-pc::x-dumb]
      (atomic macro) expand function at current subterm, without simplifying
```

ACL2 PB: Theory Management

When proofs become large and complex, theory management is key.

```
[ACL2-pc::in-theory]
    (primitive) set the current proof-builder theory
```

An ACL2 *theory* is the set or *runes* currently active.

When managing a large proof, it is often quite helpful to control what definitions may be expanded (automatically) and what rewrite rules are enabled.

This is less of an issue when using the ACL2 Proof Builder, due to the low-level, manual mechanisms that the PB provides.

But for large proofs, theory management is critical.

We will return to this issue later...

ACL2 PB: Session Management

There are commands to manage an overall Proof Builder session.

```
[ACL2-pc::comm]
    (macro) display instructions from the current interactive session

[ACL2-pc::exit]
    (meta) exit the interactive proof-builder

[ACL2-pc::replay]
    (macro) replay one or more instructions

[ACL2-pc::restore]
    (meta) remove the effect of an UNDO command

[ACL2-pc::retrieve]
    (macro) re-enter the proof-builder

[ACL2-pc::runes]
    (macro) print the runes (definitions, lemmas, ...) used

[ACL2-pc::save]
    (macro) save the proof-builder state (state-stack)

[ACL2-pc::undo]
    (meta) undo some instructions
```

ACL2 PB: Usage Advice

Although we have presented quite a few commands, it is recommended to start with a small number of commands – and add commands incrementally.

The key thing is to have clear in your mind what you are trying to do.

Remember, the Proof Builder is a record-keeping, proof-replay mechanism.

Performing hand proofs – so you know what you want to do with the Proof Builder – will help you use the Proof Builder.

After you build up your intuition about ACL2 proofs, you will use the ACL2 Theorem Prover in more and more automatic ways.