

Announcements  
-----

- Exam 2 is on Thursday
- Reading: sections 3.1-3.6, 4.1-4.2 of the book

Last time: REPL, quote, if

Today: From testing to theorem proving  
-----

But, first let and let\*:

Consider the expression:

```
(append (append '(1 2 3) '(3 4))
         (append '(1 2 3) '(3 4)))
```

Note the repetitions.

Let:

A let expression:

```
(let ((v1 x1)
      ...
      (vn xn))
    body)
```

binds its local variables, the  $v_i$ , in parallel, to the values of the  $x_i$ , and evaluates its body.

Examples:

```
(let ((x '(1 2 3))
      (y '(3 4)))
    (append (append x y) (append x y)))
```

This saves us having to type '(1 2 3) and '(3 4) multiple times.

Maybe we can avoid having to type (append x y) multiple times. What about?

```
(let ((x '(1 2 3))
      (y '(3 4))
      (z (append x y)))
    (append (append x y) z))
```

It doesn't work. What does '(1 2 3) evaluate to at the top level? '(3 4)? (append x y)? Let binds in parallel, so x and y in z binding are not yet bound.

This brings us to Let\*:

```
(let* ((v1 x1)
       ...
       (vn xn))
    body)
```

binds its local variables, the  $v_i$ , sequentially, to the values of the  $x_i$ , and evaluates its body.

```
(let* ((x '(1 2 3))
```

```
(y '(3 4))
(z (append x y))
(append (append x y) z)
```

Let's simplify further:

```
(let* ((x '(1 2 3))
       (y '(3 4))
       (z (append x y)))
      (append z z))
```

So, let, let\* give us abbreviation power.

-----

Proving theorems

Let's recall some simple algebra.

Conjecture:  $x \leq xy$  if  $y \geq 1$

What's a conjecture? A mathematical statement whose truth is under consideration.

Discussion: What does the above conjecture mean, anyway?

It means that for any values of  $x$ , and  $y$ , ... .

Really? Any values? What if  $x$  and  $y$  are functions or strings or ...? Usually the domain is implicit, i.e., "clear from context".

We will be using ACL2, and we can't appeal to "context". This is a good thing!

Notice also that can use ACL2, a programming language, to make mathematical statements. Duh! Programming languages are mathematical structures and you reason about programs the way you reason about the natural numbers, the reals, sets, etc.: you prove theorems.

In ACL2, we have to be precise about the conditions under which we expect the conjecture to hold. The conjecture can be formalized in ACL2 as follows:

```
(...
 (implies (and (integerp x)
               (integerp y)
               (>= y 1))
          (<= x (* x y))))
```

In standard mathematical notation it is:

$\langle \text{forall } x, y: x, y \in \mathbb{Z} \text{ and } y \geq 1 : x \leq xy \rangle$

Is the above conjecture true?

Well, when given a conjecture, we can try one of two things:

1. Try to falsify it.
2. Try to prove it is correct.

How do we falsify a conjecture?

Simple exhibit a counterexample.

Remember that in the design recipe, we construct examples and tests. You should do the same thing with conjectures. That is, we can test that the conjecture is true on examples. Here are some:

```
x=0,y=0
x=12,y=1
x=9,y=3/2
```

Any others?

How do we test this in ACL2s? Put the conjecture in the body of a let.

```
(let ((x 0)
      (y 0))
  (implies (and (integerp x)
                (integerp y)
                (>= y 1))
           (<= x (* x y))))
```

We are using a programming language, so we can do better. We can write a program to test the conjecture on a large number of cases. How many cases are there? We can use a random number generator to "randomly" sample from the domain.

We'll see how to do that in ACL2s.

If all of the tests pass, then we can try to prove that the conjecture is a theorem.

What would a "proof" of the above conjecture look like?

Most proofs are informal and it takes a long time for students to understand what constitutes an informal proof. This happens by osmosis over time.

In our case, we have a simple rule: it's a proof if ACL2 says it is.

```
(thm
  (implies (and (integerp x)
                (integerp y)
                (>= y 1))
           (<= x (* x y))))
```

Of course, this isn't a theorem.

Let's consider another example:

Conjecture:  $x(y+z) = xy + xz$

How do we write this in ACL2s?

```
(thm (= (* x (+ y z))
        (+ (* x y) (* x z))))
```

Wait a minute what about the assumptions, e.g., that  $x$ ,  $y$ , and  $z$  are integers?

Is the above conjecture true?

Well, we can try to falsify it.

```
(let ((x 0)
```

```
(y 0)
(z 0))
(= (* x (+ y z))
  (+ (* x y) (* x z))))
```

We can try many examples. We can automatically generate random examples? When do we give up falsifying this?

Can we just try all the possibilities? If we had infinite time. Do we? Maybe (ask a physicist), but, as a practical matter, we currently don't.

Maybe we should consider a proof. Can we prove the above?

One answer might be: "of course, multiplication distributes over addition".

In ACL2, the conjecture turns out to be true

```
(thm (implies (and (integerp x)
                  (integerp y)
                  (integerp z))
              (= (* x (+ y z))
                (+ (* x y) (* x z)))))
```

This is pretty amazing because a proof gives us a finite way of running an infinite number of examples. That's the power of logic and mathematics.

In fact, the conjecture turns out to be true without restrictions on the domain of  $x$ ,  $y$ , and  $z$ .

```
(thm (= (* x (+ y z))
        (+ (* x y) (* x z))))
```

What that means is that it is true for atoms (numbers, strings, symbols, characters, etc) and conses.

You will find that you can often prove stronger theorems than you need, especially if you define your functions using the appropriate idioms.

Why? Remember that what we try to do is to coerce objects outside the intended domain to "unit" elements in the intended domain, so often theorems hold on the whole universe. A nice consequence of totality.

When ACL2 proves this theorem, is it thinking?

The question of whether Machines Can Think ... is about as relevant as the question of whether Submarines Can Swim.  
EWD898, 1984

See the EWD archives at the University of Texas at Austin.

Conjecture1:  $x+(yz) = (x+y) (x+z)$

Let's try some examples:

```
x=0,y=1,z=2
x=3,y=0,z=0
x=1/3,y=1/3,z=1/3
```

Try it with:

```
(let ((x ..)
      (y ..)
      (z ..))
  (= (+ x (* y z))
     (* (+ x y) (+ x z))))
```

Is it true?

```
(thm (= (+ x (* y z))
        (* (+ x y) (+ x z))))
```

So, let's try to simplify:

```
      x+yz = xx + xz + xy + yz
iff  x = xx + xz + xy
iff  1 = x+z+y
```

So, if  $x+y+z \neq 1$ , then the above doesn't hold?

Right?

Wrong. What allowed us to divide by  $x$ ? What if  $x$  is 0?  
In fact, look at the first example we tried.

Look at what ACL2s produces. (It knows not to divide by  $x$ .)

WRAP UP  
-----

In these 3 cases, it was easy to decide if a conjecture was true or false, and with a good amount of testing, we would have identified the false conjectures.

Is this always the case?

No.

Anyone heard of Fermat's last theorem?

For all positive integers  $x$ ,  $y$ ,  $z$ , and  $n$ , where  $n > 2$ ,  
 $x^n + y^n \neq z^n$

In 1637, Fermat wrote about the above:

"I have a truly marvelous proof of this proposition which this margin is too narrow to contain."

This is called Fermat's Last Theorem. It took 357 years for a correct proof to be found (by Andrew Wiles in 1995).

Can someone use the above to construct a conjecture that would be hard to prove in ACL2?

```
(defun f (x y z n)
  (if (and (posp x)
          (posp y)
          (posp z)
          (natp n)
          (> n 2)
          (= (+ (expt x n) (expt y n))
             (expt z n))))
      1
      0))
```

```
(thm (= (f x y z n) 0))
```

So, proving theorems may be hard.

But, if they aren't theorems, we should be able to find counterexamples quickly, right?

No.

A conjecture may be false, but it may be *very* hard to find a counterexample.

Let  $Z_2$  be the set of integers  $\geq 2$ . Remember that we can factor any integer in  $Z_2$  into a product of primes. This product is uniquely defined.

If the number of primes is even, we say that the number is of "even type"; otherwise it is of "odd type".

Examples:  $2=2$  is of odd type and  $4=2*2$  is of even type.

Let  $E(n)$  = the number of integers in  $Z_2$  that are of even type and are  $\leq n$ .

Let  $O(n)$  = the number of integers in  $Z_2$  that are of odd type and are  $\leq n$ .

Polya's conjecture:  $O(n) > E(n)$  for all  $n$  in  $Z_2$ .

This conjecture was made in 1919.

It was checked for many values of  $n$ , and it was widely believed to be true.

However, in 1962 a counterexample was found (by Lehman) for  $n=906,180,359$ .

This is related to Computer Science.

Perhaps the most well-known statement explaining how is:

Program testing can be used to show the presence of bugs, but never to show their absence! EWD249, 1970

Is this really a problem? Yes. Floating point & AMD.

But, eventually, we can resolve all such problems, right?

Who knows? There are open conjectures dating back to the ancient Greeks.

Actually, we do know, from Godel that there are "true theorems" that are not provable, but that will have to wait until you take a more advanced class on logic.