Definitions and Proving Equality Formulas

Last lecture, we introduced the ACL2 logic, and showed how it could be used to reason about some simple equalities, including when those equalities talked about functions applying to values.

One thing we did not do, however, is take advantage of special knowledge we may have of those functions. For instance, we know that the function `car` does interact with `cons`, if only because we can check that `(car (cons 1 2))` is the same as `1`. We remedy this situation here.

To every primitive function (or special form) in ACL2, there correspond *axioms* that capture how those functions behave. Recall from last time that an axiom is a formula that is taken by fiat to be valid. There are a lot of axioms in ACL2, and let me list some here. (We'll introduce more as the course progresses, when needed. I'll try to prepare a handout listing the most common ones.) Note that some of these axioms can be presented in a couple of ways—I will chose one way now, and reserving myself the right to present some of these axioms differently later on. I also make no claim of minimality; some of these axioms are redundant.

if axioms	`(= (if NIL x y) y)`
	`(= (if` v `x y) x)` for every value v not `NIL`
	`(= (if (cons a b) x y) x)`
nil axiom	`¬x` $\equiv$ `(= x NIL)`
endp axioms	`(endp` v`)` for every value v
	`¬(endp (cons a b))`
	`(endp x)` $\equiv$ `(= (endp x) T)`
consp axioms	`(consp x)` $\equiv$ `¬(endp x)`
	`(consp x)` $\equiv$ `(= (consp x) T)`
car axioms	`(= (car (cons x y)) x)`
	`(endp x)` $\Longrightarrow$ `(= (car x) NIL)`

cdr axioms	`(= (cdr (cons x y)) y)`
	`(endp x)` $\Longrightarrow$ `(= (cdr x) NIL)`
cons axiom	`(consp x)` $\Longrightarrow$ `(= (cons (car x) (cdr x)) x)`

For arithmetic operations, we have a bunch of axioms, some of which include:

arithmetic axioms	`(= (+ x y) (+ y x))`
	`(= (+ x 0) x)`
	`(= (* x y) (* y x))`
	`(= (* x 0) 0)`
	`(= (* x 1) x)`
	`(= (* x (+ y z)) (+ (* x y) (* x z)))`
	`(>= x y)` $\wedge$ `(>= y z)` $\Longrightarrow$ `(>= x z)`
	...

Basically, whenever you have an equation or inequation involving `=`,`>=`,`+`,`*`,`expt`, `-`, we will assume that you can do the same kind of arithmetic reasoning that you've done since high school, and establish whether the equation is true or false for all numbers. We will therefore assume we have a bunch of axioms, corresponding to the valid formulas of arithmetic, and call that axiom, collectively, *arithmetic reasoning*.

Okay, the above takes care of (some) of the primitive expressions.

What about user-defined functions, the ones introduced by `defun`? Glad you asked. Basically, every defun of the form `(defun foo (x y) BODY)` introduces a new axiom in the system, of the form `(= (foo x y) BODY)`. The axiom just says that applying the function to arguments is equal to the body of the function. Exactly what you would expect.

Here are some examples. Here is the append function, which I call `app` here because `append` is already predefined in ACL2.

```
(defun (app x y)
  (if (endp x)
      y
      (cons (car x) (app (cdr x) y))))
```

(Note that I have written the function using `if` instead of `cond`—reasoning about `if` is much easier than reasoning about `cond`. Why do you think?)

This definition gives rise to a new axiom that you can use:

```
(= (app x y) (if (endp x) y (cons (car x) (app (cdr x) y))))
```

And that's it. Remember this: *Every definition gives rise to a new axiom.*

Many of the axioms above have the form `(= exp exp')`, and it turns out that many of the theorems we will prove also have this pretty simple form.

Here is a new Equational Proof Rule that you can use in your equational proofs, that will make it easier to use theorems (including axioms) of that form. The rule tells you that if you have a theorem of the form `(= exp exp')`, then you can always rewrite a formula by replacing some `exp` in the formula into `exp'`—because, after all, they are equal. Formally, here is the rule:

> **Rule of Equality Substitution 1**: If `(= exp exp')` is an instance of a theorem, then we can rewrite a formula F by replacing occurrences of `exp` in F by `exp'`.

Let's look at an easy example of the proof of a theorem of the form `(= exp exp')`. What happens when we cons a value in front of a list created by appending two lists together? It should be the same as consing the value in front of the first of the two lists, and then appending, right? Think about it. We can prove it, as follows.

Theorem 1. `(= (cons a (app x y)) (app (cons a x) y))`

Proof.

```
(= (cons a (app x y))
   (app (cons a x) y))
```

by Substitution with definition of ***app***

```
(= (cons a (app x y))
   (if (endp (cons a x)) y (cons (car (cons a x)) (app (cdr (cons a x)) y))))
```

by Substitution with endp axiom

```
(= (cons a (app x y))
   (if NIL y (cons (car (cons a x)) (app (cdr (cons a x)) y))))
```

by Substitution with if axiom

```
(= (cons a (app x y))
   (cons (car (cons a x)) (app (cdr (cons a x)) y)))
```

by Substitution with car axiom

```
(= (cons a (app x y))
   (cons a (app (cdr (cons a x)) y)))
```

by Substitution with cdr axiom

```
(= (cons a (app x y))
```

```
   (cons a (app x y)))
```

which is an instance of reflexivity for `=`

□

That's long, and I've done all the steps very slowly, pointing out exactly what I am doing. *Make sure you understand all the steps in the above proof.* In fact, we can probably do a bunch of substitutions together, and because *Substitution with* is so common, I will often leave it out. I will ask you to indicate all the axioms and definitions you are using, though, so that an acceptable alternate proof might look like:

```
(= (cons a (app x y)) (app (cons a x) y))
```

by definition of `app`

```
(= (cons a (app x y))
   (if (endp (cons a x)) y (cons (car (cons a x)) (app (cdr (cons a x)) y))))
```

by endp axiom and if axiom

```
(= (cons a (app x y)) (cons (car (cons a x)) (app (cdr (cons a x)) y)))
```

by car axiom and cdr axiom

```
(= (cons a (app x y)) (cons a (app x y)))
```

by reflexivity of `=`

Here's another easy one: what happens when you append `NIL` in front of another list? It should give back the original list, right? Try it out on some values: `(app NIL '(1 2 3))` yields `(1 2 3)`, etc.

Let's prove it.

Theorem 2. `(= (app NIL y) y)`

Proof.

```
(= (app NIL y) y)
```

by definition of `app`

```
(= (if (endp NIL) y (cons (car NIL) (app (cdr NIL) y))) y)
```

by endp and if axioms

```
(= y y)
```

by reflexivity of `=`

□

There. An almost trivial proof.

What about the other way around? Could we try to prove that `(= (app x NIL) x)`? Actually, is that even true? Can you come up with a counter example, that is, a value for `x` such that `(app x NIL)` is not equal to `x`? Hint: there are many! Because we can find a counter example, we know the original formula is not valid, so we don't even bother trying to prove it. It's bound to fail anyways.