Pete Manolios, CSU290 Lecture Notes, Mon Feb 11 2008

Announcements

- HW3 is up and due next Monday at 10 AM.

- You *have to* work in pairs for this homework and you have to work with someone you have not worked with yet. Send the name of your partner to Zhifeng (austin@ccs.neu.edu) by the end of the day, on Wednesday Feb. 13th. If you do not do this, you will get a 0 on your homework.

- Each pair should submit their homework via blackboard. Each pair should submit just once. That is, one person per team should submit the homework via their blackboard account. Your submission should be a text file. The format of the file should be:

  - the names of the team

  - clear solutions to the problems

- Reading for this part of course: 6.1-6.2,6.4

Some observations from exam 2:

Let’s look at some examples of things people wrote:

```
(defun foo (x y)
  ... (foo x (cons _ y)) ...)
```

This will probably never terminate, and what design recipe will lets you do this?

```
; foo: true-listp -> true-listp
(defun foo (l)
  (cond ((true-listp l)
         (cond ...)
         (t ...))))
```

Assume that you have a true-list! (I’ll have to keep repeating it until you get it).

Why did so many of you write (prefixes ’()) = ’() ???

Here is what I wrote on the board: (prefixes ’()) = ’(())!!!

```
(defun prefixes (l)
  ... (list (prefixes (rest l)) (remlast l) l) ...)
```

Always returns a list of length 3.

```
(defun remove-last (x)
  (cond ((= (len x) 1) nil)
        (t (cons (car x) (remove-last (cdr x))))))
```

Non-terminating. The design recipe!!!

```
(defun prefixes (l)
  (cond (( ... ))
        (t ... (append ... (prefixes l) ... ))))
```

Non-terminating. The design recipe!!!

Review

Axioms:

- Every propositional tautology is an axiom.
- Reflexivity: x = x
- Equality Axiom Schema for Functions:
  For every function symbol f of arity n we add the axiom
  x1 = y1 /\ ... /\ xn = yn => (f x1 ... xn) = (f y1 ... yn)
- Transitivity of Equality:
  x = y /\ y = z => x = z

- x = y => (equal x y) = t
- x != y => (equal x y) = nil

- x = nil => (if x y z) = z
- x != nil => (if x y z) = y

- (equal (car (cons x y)) x) != nil
- (cdr (cons x y)) = y
- (consp (cons x y)) = t

... for other data types

For example:

(implies (consp x) (not (symbolp x)))

which together with
(symbolp nil)

implies that (consp nil) = nil.

ACL2 Logic, continued

The last rule of inference is instantiation:
Derive \phi / \sigma from \phi

Example: From (equal (car (cons x y)) x)
I can derive (equal (car (cons (foo x) (bar z))) (foo x))

More carefully, a substitution is just a list of the form:
((var1 term1) ... (varn termn)), where the vars are "target variables" and the terms are their images. The application of this substitution to a formula uniformly replaces every free occurrence of a target variable by its image.

We already have enough of a theory developed to prove some theorems, e.g.:

t != nil

Proof:
Proof Sketch. We have an axiom telling us (equal (car (cons a b)) a) /= nil. Call this ‘‘line 1’’ of our proof. Using line 1 and the two axioms about equal we can prove that (equal (car (cons a b)) a) = t. Then, using substitutions of equals for equals, substitute t for (equal (car (cons a b)) a) in line 1 to derive t /= nil. Q.E.D.

Here is its formalization.

```
Proof:
1. (equal (car (cons a b)) a) /= nil        {Axiom relating car, cons, equal, instantiation}
2. (car (cons a b)) /= a -> (equal (car (cons a b)) a) = nil
                                            {Equal axiom, instantiation}
3. ~((car (cons a b)) /= a)                 {Modus Tollens, 1,2}
4. ~(~((car (cons a b)) = a))               {Abbrev, 3}
5. (car (cons a b)) = a                     {Doub Neg ~~f <-> f,  4}
6. (car (cons a b)) = a -> (equal (car (cons a b)) a) = t
                                            {Equal Axiom}
7. (equal (car (cons a b)) a) = t           {Modus Ponens, 5,6}
8. t /= nil                                 {Subst Equality Axiom, 1,7}
Q.E.D.
```

BTW, if our last axiom was

```
(equal (car (cons x y)) x) = t,
```

we would not be able to prove this theorem. Why?

We’re not going to be this pedantic. We are going to operate at a higher level, adopting a rigorous, but informal notion of proof, but one that is formalizable (i.e., ACL2 checkable). Let’s consider some examples.

First, we are given that

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

In fact, every time we define a function, we get an axiom of the form (f x1 ... xn) = body. More on that later.

Let’s prove a few theorems.

```
1. (app (cons x y) z)
 = (cons x (app y z))
```

Note, I will allow myself to write above instead of

```
(equal (app (cons x y) z)
       (cons x (app y z)))
```

```
  (app (cons x y) z)
=   {Definition of app, instantiation}
  (if (endp (cons x y))
      z
    (cons (car (cons x y)) (app (cdr (cons x y)) z)))
=   {Definition of endp, axioms consp}
  (if nil
      z
    (cons (car (cons x y)) (app (cdr (cons x y)) z)))
=   {Axioms for if, car, cdr}
  (cons x (app y z))
```