Logic and Computation	Lecture 20
CSU 290 Spring 2009 (Pucella)	Thursday, Mar 12, 2009

Note that I change the name of the functions slightly in these notes from what I used in class, to be consistent with what I did in Lecture 6 when talking about tail recursion.

Generalization

Today, we nail down the last few bits we need to prove some of the claims we made a while back, and some of the claims that have been made in 211, about tail-recursive functions.

Consider the function `sum` that sums all the items in a list:

```
(defun sum (L)
  (if (endp L)
      0
    (+ (car L) (sum (cdr L)))))
```

Suppose you wanted to write this function tail-recursively, that is, with an accumulator. The resulting function `sum-acc`, following what you learned in 211 and adapted to our design recipe as we saw earlier in the course:

```
(defun sum-aux (L acc)
  (if (endp L)
      acc
    (sum-aux (cdr L) (+ (car L) acc))))

(defun sum-acc (L)
  (sum-aux L 0))
```

We expect that `(= (sum x) (sum-acc x))`—that's the point, really, that the tail-recursive version is the same as the recursive version.

Let's try to prove it then. By substitution, the formula is equivalent to `(= (sum x) (sum-aux x 0))`. To prove that, because we know nothing about `x` and because `sum` (and `sum-aux`) are recursive and controlled by `x`, it calls for doing induction on `x`. Here are the proof obligations:

P1: `(endp x)` $\Longrightarrow$ `(= (sum x) (sum-aux x 0))`

P2: $\neg$`(endp x)` $\wedge$ `(= (sum (cdr x)) (sum-aux (cdr x) 0))`
$\Longrightarrow$ `(= (sum x) (sum-aux x 0))`

P1 is trivial to prove, so I won't bother here. (Do it if you are still shaky.) P2 is more interesting. The context has two assumptions in it:

A1: ¬`(endp x)`

A2: `(= (sum (cdr x)) (sum-aux (cdr x) 0))`

Let's prove the consequent of the implication:

`(= (sum x) (sum-aux x 0))`

by def of ***sum****, A1, if axiom*

`(= (+ (car x) (sum (cdr x))) (sum-aux x 0))`

by def of ***sum-aux****, A1, if axiom*

`(= (+ (car x) (sum (cdr x))) (sum-aux (cdr x) (+ (car x) 0)))`

by A2

`(= (+ (car x) (sum-aux (cdr x) 0)) (sum-aux (cdr x) (+ (car x) 0)))`

by arithmetic

`(= (+ (car x) (sum-aux (cdr x) 0)) (sum-aux (cdr x) (car x)))`

And now we're stuck. There's really nothing we can use. We could try to prove a lemma to get us out of the jam, but let's try something else instead. Let's try to generalize what we're trying to prove.

Part of the problem, why we get stuck during the proof, is that we have no information about `sum-aux` when the second argument is not 0—the inductive hypothesis A2 is useless, and it came from the theorem we were trying to prove. Can we prove something more general, that can deal with the case where the second argument to `sum-aux` is not 0? We could `(sum-aux x y)` be equal to, in terms of `(sum x)`? It doesn't take long to convince yourself that `(= (sum-aux x y) (+ (sum x) y))` is what we want. Let's try to prove that then. Again, we proceed by induction on `x`, and we get the following proof obligations:

P1: `(endp x)` $\Longrightarrow$ `(= (sum-aux x y) (+ (sum x) y))`

P2: ¬`(endp x)` $\wedge$ `(= (sum-aux (cdr x) y) (+ (sum (cdr x)) y))`
$\Longrightarrow$ `(= (sum-aux x y) (+ (sum x) y))`

Again, P1 is trivial to prove, so I won't bother here. P2 is more interesting. The context has two assumptions in it:

A1: ¬`(endp x)`

A2: `(= (sum-aux (cdr x) y) (+ (sum (cdr x) y))`

Let's prove the consequent of the implication:

```
(= (sum-aux x y) (+ (sum x) y))
```

by def of **sum**, *A1, if axiom*

```
(= (sum-aux x y) (+ (+ (car x) (sum (cdr x))) y))
```

by def of **sum-aux**, *A1, if axiom*

```
(= (sum-aux (cdr x) (+ (car x) y)) (+ (+ (car x) (sum (cdr x))) y))
```

by commutativity and associativity of **+**

```
(= (sum-aux (cdr x) (+ (car x) y)) (+ (sum (cdr x)) (+ (car x) y)))
```

Which is tantalizing clos to being provable, but isn't given what we have. Notice that we cannot apply A2: it talks about `(sum-aux (cdr x) y)`, but what we have is `(sum-aux (cdr x) (+ (car x) y))`. We cannot perform substitution in an assumption in the context—that's not allowed, because it could lead to unsoudness (being able to prove false). This is something we never really talked about, because the issue never really arose, but that's the case nevertheless. So we're stuck, we cannot apply A2.

So what *can* we do? It turns out that there's a way out of the jam by relying on a feature of the induction rule that I did not really point out back then, although I left the door open for it. Recall the induction rule from Lecture 17:

> **Induction Rule (for true lists)**: If F is a formula, `x` a variable in F whose intended domain is true lists, and σ is a substitution that, among others, sends variable `x` to `(cdr x)`, then we can rewrite a formula F into
>
> $$\texttt{(endp x)} \Longrightarrow F \quad \wedge \quad (\neg\texttt{(endp x)} \wedge F/\sigma) \Longrightarrow F$$
>
> where F/σ is instance of formula F obtained by performing the substitution σ.

Note that σ can be *any* substitution, as long as it sends `x`, the induction variable, to `(cdr x)`. In particular, we are free to send `y` to any expression when we create the proof obligations for the induction. In other words, when we apply the induction rule, we get to tailor the inductive hypothesis as we want, as long as we do so by substituting for the non-induction variables. (Clearly, we need to substitute every occurrence of the same variable by the same expression.) In the case of the above proof, clearly, we want to substitute `(+ (car x) y)` for `y`, so let's do that. Now we can push the full proof through.

Theorem 1. `(= (sum-aux x y) (+ (sum x) y))`

Proof. By induction on `x`, and using a substitution that sends `y` to `(+ (car x) y)`, we get the following proof obligations:

P1: `(endp x)` $\Longrightarrow$ `(= (sum-aux x y) (+ (sum x) y))`

P2: $\neg$`(endp x)`
$\wedge$ `(= (sum-aux (cdr x) (+ (car x) y)) (+ (sum (cdr x)) (+ (car x) y)))`
$\Longrightarrow$ `(= (sum-aux x y) (+ (sum x) y))`

P1 is trivial to prove, so I won't bother here. P2 is more interesting. The context has two assumptions in it:

A1: $\neg$`(endp x)`

A2: `(= (sum-aux (cdr x) (+ (car x) y)) (+ (sum (cdr x) (+ (car x) y))))`

Let's prove the consequent of the implication:

`(= (sum-aux x y) (+ (sum x) y))`

by def of ***sum****, A1, if axiom*

`(= (sum-aux x y) (+ (+ (car x) (sum (cdr x))) y))`

by def of ***sum-aux****, A1, if axiom*

`(= (sum-aux (cdr x) (+ (car x) y)) (+ (+ (car x) (sum (cdr x))) y))`

by commutativity and associativity of `+`

`(= (sum-aux (cdr x) (+ (car x) y)) (+ (sum (cdr x)) (+ (car x) y)))`

by A2

`(= (+ (sum (cdr x) (+ (car x) y)) (+ (sum (cdr x)) (+ (car x) y)))`

by reflexivity of `=`

□

Now that we have Theorem 1, we can easily prove the original formula we wanted to prove:

Theorem 2. `(= (sum x) (sum-acc x))`

Proof.

`(= (sum x) (sum-acc x))`

by def of ***sum-acc***

```
(= (sum x) (sum-aux x 0))
```

by Theorem 1

```
(= (sum x) (+ (sum x) 0))
```

by arithmetic

```
(= (sum x) (sum x))
```

by reflexivity of =

□

This pattern occurs most times when you try to prove the equivalence of a recursive and a tail-recursive version of a function. For instance, consider `rev`, and the tail-recursive version `rev-acc`, using an auxiliary function `rev-aux`—see Lecture 6, but be aware I messed up the name of `rev-acc` in the notes, calling it simply `rev`.

The theorem we want is that `(= (rev x) (rev-acc x))`, which is just `(= (rev x) (rev-aux x NIL))`. As above, if you try to prove it directly, you get stuck. Generalization helps, by letting us prove something about `(rev-aux x y)` in general. The theorem we want is `(= (rev-aux x y) (app (rev x) y))`.

> **Exercise:** *Prove that theorem—pay attention to the proof obligations you need to push the proof through.*

Then, once you have that theorem in hand, you can easily prove `(= (rev x) (rev-acc x))`.

> **Exercise:** *do it.*