Logic and Computation	Lecture 15
CSU 290 Spring 2009 (Pucella)	Thursday, Feb 19, 2009

In this lecture, we round off some aspects of proving theorems, before moving on to more advanced proving techniques.

In the last two lectures, we saw some basic techniques (basically rewriting using substitutions and propositional reasoning) to prove theorems of the form `(= exp1 exp2)` and theorems of the form $F \Longrightarrow$ `(= exp1 exp2)`.[1]

Using Conditional Theorems

Up until now, to help us in our proofs, we have been using theorems and assumptions of the form `(= exp1 exp2)`, which can be applied directly. However, it is often the case that the theorems that we want to use are themselves implications, and we need to learn what to do when that is the case.

Here is an example. Assume that we have

```
(= (true-listp y) T) ⟹ (= (true-listp (app x y)) T)
```

for our standard definition of `app`. That is actually a valid formula (convince yourself of that!), but we cannot prove that just yet. So let's just assume it. (We'll be able to prove it at the end of the next week, if you're curious.) Equipped with this assumption, we prove the following theorem:

```
(= (true-listp (app-nil x)) T)
```

where `app-nil` is defined as follows:

```
(defun app-nil (x) (app x NIL))
```

The proof is fairly straightforward, but requires the assumption.

```
(= (true-listp (app-nil x)) T)
```

by definition of ***app-nil***

```
(= (true-listp (app x NIL)) T)
```

At this point, we would like to use the assumption, which says, if you'll recall, that `(= (true-listp (app x y)) T)` is true when `(= (true-listp y) T)` is true. In particular,

[1] It is not too hard to generalize the techniques we saw to theorems of the form `exp` and $F \Longrightarrow$ `exp`, simply by rewriting `exp` to any value that is not `NIL`, since any non-`NIL` value is interpreted as true.

we want to use the following instance of the assumption: `(= (true-listp NIL) T)` $\Longrightarrow$ `(= (true-listp (app x NIL)) T)`. If we can establish that `(= (true-listp NIL) T)` is true at the point where we want to use it in the proof, then we can get from it that `(= (true-listp (app x NIL)) T)` is true, and we are done. And it is easy to check that `(= (true-listp NIL) T)` is true, via a very simple proof (do it!). Thus, we can complete the proof in one step. Here is the complete proof from the beginning.

```
(= (true-listp (app-nil x)) T)
```

by definition of ***app-nil***

```
(= (true-listp (app x NIL)) T)
```

by assumption (since ***(= (true-listp NIL) T)*** *is true)*

```
(= T T)
```

by reflexivity

Actually, as I said above, we can simplify this slightly by dropping the `(= .. T)` bit, since any value which is not `NIL` corresponds to true in the ACL2 logic, so that, for instance, $5 \equiv$ *true*. Thus, we can assume simply that `(true-listp y)` $\Longrightarrow$ `(true-listp (app x y))`, and prove `(true-listp (app-nil x))` as follows:

```
(true-listp (app-nil x))
```

by definition of ***app-nil***

```
(true-listp (app x NIL))
```

by assumption (since ***(true-listp NIL)*** *is true)*

We will see slightly more complex examples of using theorems that are implications later.

Proof by Case Analysis

Consider how you would prove that `(>= (if x 20 15) 0)` is valid. It shouldn't take too long to convince yourself that it is. And your reasoning, informally, should be that `x` here is either true or false, and either way, the result is greater than 0. The question is, how can we prove that using the rules we already have?

What we want is case analysis, or proof by cases. Here is the technique, used to prove the

above:

```
(>= (if x 20 15) 0)
```

by case analysis on $\boldsymbol{x}$

$$\texttt{x} \Longrightarrow \texttt{(>= (if x 20 15) 0)} \quad \wedge \quad \neg\texttt{x} \Longrightarrow \texttt{(>= (if x 20 15) 0)}$$

And now we can prove both conjuncts separately, rather quickly:

$\texttt{x} \Longrightarrow \texttt{(>= (if x 20 15) 0)}$

by assumption $\boldsymbol{x}$ *and if axiom*

$\texttt{x} \Longrightarrow \texttt{(>= 20 0)}$

by arithmetic

$\texttt{x} \Longrightarrow \texttt{T}$

by propositional reasoning

and

$\neg\texttt{x} \Longrightarrow \texttt{(>= (if x 20 15) 0)}$

by assumption $\neg\boldsymbol{x}$ *and if axiom*

$\neg\texttt{x} \Longrightarrow \texttt{(>= 15 0)}$

by arithmetic

$\neg\texttt{x} \Longrightarrow \texttt{T}$

by propositional reasoning

(**For the curious:** Why are we allowed to do this? Well, case analysis is a "macro step" that can be understood in terms of good old propositional reasoning. (Yes, propositional reasoning packs a lot of punch.) Here is the idea.

Given a formula G to be proved, if we can identify a formula F which makes is possible to prove G when F is true and when F is false (generally, such an F will control a conditional such as `if`), then we can rewrite G as:

G

by propositional reasoning (using $true \Longrightarrow p \equiv p$*)*

$true \Longrightarrow G$

by propositional reasoning (using $p \vee \neg p \equiv true$*)*

$$(F \vee \neg F) \Longrightarrow G$$

$$\boxed{\textit{by propositional reasoning}}$$

$$(F \Longrightarrow G) \wedge (\neg F \Longrightarrow G)$$

And this is what case analysis gives you. If you choose F correctly, this technique can help. Note that you can choose *anything* for F.)

Case analysis is especially handy to prove equivalences of Boolean expressions.

In particular, remember when I flubbed the definition of `and` in ACL2 back in lecture 10, and I claimed that the two definitions I gave in fact were equivalent? Here is a proof, using case analysis.

The definition of `and` is as follows:

```
(defun and (x y)
  (if x y x))
```

Here is an "alternate" definition:

```
(defun alt-and (x y)
  (if x y NIL))
```

I claim that the following is valid `(= (and x y) (alt-and x y))`, and here is a proof:

`(= (and x y) (alt-and x y))`

by definition of ***and*** *and* ***alt-and***

`(= (if x y x) (if x y NIL))`

by case analysis on ***x***

`x` $\Longrightarrow$ `(= (if x y x) (if x y NIL))` $\wedge$ $\neg$`x` $\Longrightarrow$ `(= (if x y x) (if x y NIL))`

Let's prove the two conjuncts separately:

`x` $\Longrightarrow$ `(= (if x y x) (if x y NIL))`

by assumption ***x*** *and if axioms*

`x` $\Longrightarrow$ `(= y y)`

by reflexivity of `=` *and propositional reasoning*

and

$$\neg \texttt{x} \Longrightarrow \texttt{(= (if x y x) (if x y NIL))}$$

by nil axiom

$$\texttt{(= x NIL)} \Longrightarrow \texttt{(= (if x y x) (if x y NIL))}$$

by assumption `(= x NIL)` *and if axioms*

$$\texttt{(= x NIL)} \Longrightarrow \texttt{(= NIL NIL)}$$

by reflexivity of `=` *and propositional reasoning*

In fact, let's settle something that may have left you wondering a bit. (Or maybe not; who knows?) As you may have noticed, we can write a formula $\texttt{exp} \wedge \texttt{exp'}$, but also `(and exp exp')`. In homeworks 2 and 3, we have essentially been using those interchangeably. Does that make sense? After all, they are different formulas, if only because they are written differently: the first is a conjunction of two ACL2 expressions, while the second is a single ACL2 expression, which in fact corresponds to a big `if` expression. The reason why we can flip between the two without batting an eye is that they are in fact equivalent expressions, as the following theorem shows:

Theorem 1. $\texttt{a} \wedge \texttt{b} \equiv \texttt{(and a b)}$

Proof. By case analysis on `a`, $\texttt{a} \wedge \texttt{b} \equiv \texttt{(and a b)}$ is equivalent to

$$(\texttt{a} \Longrightarrow (\texttt{a} \wedge \texttt{b} \equiv \texttt{(and a b)})) \quad \wedge \quad (\neg \texttt{a} \Longrightarrow (\texttt{a} \wedge \texttt{b} \equiv \texttt{(and a b)}))$$

and we prove the two conjuncts separately.

The first conjunct:

$$\texttt{a} \Longrightarrow (\texttt{a} \wedge \texttt{b} \equiv \texttt{(and a b)})$$

by definition of `and`

$$\texttt{a} \Longrightarrow (\texttt{a} \wedge \texttt{b} \equiv \texttt{(if a b a)})$$

by assumption `a` *and if axiom*

$$\texttt{a} \Longrightarrow (\texttt{a} \wedge \texttt{b} \equiv \texttt{b})$$

by propositional reasoning, since $p \Longrightarrow (p \wedge q \equiv q)$ *is valid*

And the second conjunct:

$\neg \mathtt{a} \Longrightarrow (\mathtt{a} \wedge \mathtt{b} \equiv \mathtt{(and\ a\ b)})$

by definition of ***and***

$\neg \mathtt{a} \Longrightarrow (\mathtt{a} \wedge \mathtt{b} \equiv \mathtt{(if\ a\ b\ a)})$

by assumption $\neg \mathtt{a}$ *and if axiom*

$\neg \mathtt{a} \Longrightarrow (\mathtt{a} \wedge \mathtt{b} \equiv \mathtt{a})$

by propositional reasoning, since $\neg p \Longrightarrow (p \wedge q \equiv p)$ *is valid*

□

It turns out this is the case for all Boolean operations in ACL2. Recall the definitions:

```
(defun or (x y)
  (if x x y))

(defun not (x)
  (if x NIL T))

(defun implies (x y)
  (if x (if y T NIL) T))

(defun iff (x y)
  (if x (if y T NIL) (if y NIL T)))
```

Exercise: prove the following theorems

$(\mathtt{a} \vee \mathtt{b}) \equiv \mathtt{(or\ a\ b)}$

$(\neg \mathtt{a}) \equiv \mathtt{(not\ a)}$

$(\mathtt{a} \Longrightarrow \mathtt{b}) \equiv \mathtt{(implies\ a\ b)}$

$(\mathtt{a} \equiv \mathtt{b}) \equiv \mathtt{(iff\ a\ b)}$