Logic and Computation **Lectures 12**
CSU 290 Spring 2009 (Pucella) **Monday, Feb 9, 2009**

Reasoning about Equality and Function Symbols

Last time, we saw a generalization of propositional logic to reason about equality of values. The goal today is to generalize this somewhat, to eventually get to the full ACL2 logic, which is a form of quantifier-free first-order logic of the kind I pointed out last time.

Let me restart the presentation from scratch, and built up the right logic with the right terminology.

First, some motivation. Ultimately, we want to reason about ACL2 expressions: At the end of the day, we want to be able to state very general properties of programs written in ACL2.

For instance, we can check by testing that for the definition of factorial `fact` we gave a couple of lectures back, `(fact n)` seems always positive: `(fact 1)` is, as is `(fact 2)`, as is `(fact 3)`. It seems quite reasonable to think that `(fact n)` is positive for any `n`. But of course testing does not tell us that, cannot tells us that. (Why not?)

What we will develop is a logic that will let us *prove* that for all choices of `n`, `(fact n)` is positive. That is, for all `n`, `(>= (fact n) 0)` is true. In fact, we will be able to say more. Further testing may suggest that `fact` always returns an integer. And indeed, we will be able to prove that `(fact n)` is always an integer, that is, that for all `n`, `(integerp (fact n))` is true. In fact, we will be able to prove that for all `n`, `(>= (fact n) 0)` $\wedge$ `(integerp (fact n))` is true.

So, let's extend the logic we introduced last week, and let's call it the *ACL2 logic*. I will use $F, G, H, \ldots$ to range over formulas of ACL2 logic to distinguish them from propositiional logic formulas.

A formula F, G of ACL2 logic is one of:

- *true*, *false* (constants)
- `exp` (any expression in the ACL2 language)
- $\neg F$ (negation)
- $F \wedge G$ (conjunction)
- $F \vee G$ (disjunction)
- $F \Rightarrow G$ (implication)

- $F \equiv G$ (equivalence, also written $F \Leftrightarrow G$).

Thus, a formula F looks just like a propositional formula, except that it has ACL2 expressions instead of propositional variables.

So, a formula such as above, `(>= (fact n) 0)` $\wedge$ `(integerp (fact n))`, is a formula of ACL2 logic.

We care about the *validity* of such formulas, which in analogy with validity for propositional logic formulas, means that for all choices of values for the variables in the formula, the resulting formula is true. The truth conditions for the logical connectives are given by the usual truth tables, and the truth conditions for the basic expressions of ACL2 are as we saw in the "Propositional Logic in ACL2" lecture: any value that is not `NIL` is considered true, while `NIL` is considered false.

To prove validity of such formulas, we cannot use a truth table or exhaustively try all possible value, so we must rely on an indirect method such as an equational proof. An equational proof for the ACL2 logic is just like one for propositional logic, but we will refine the definition somewhat. As a matter of terminology, a formula for which you can find an equational proof we will call a *theorem*.

First off, we need some useful concepts.

An *instance of a propositional formula* is a propositional formula in which we have replaced every propositional variable by a formula of ACL2 logic (the same formula replacing the same propositional variable, of course). Thus, (`(= x y)` $\wedge$ `(= y z)`) $\Rightarrow$ `(= x y)` is an instance of the propositional formula $(p \wedge q) \Rightarrow p$, where p has been replaced by `(= x y)` and q has been replaced by `(= y z)`.

Similarly, an *instance of an ACL2 formula* is an ACL2 formula in which we have replaced every variable by an expression of ACL2. (Since variables are expressions in ACL2, we can replace a variable by another variable, of course.) Thus, (`(= a (+ b c))` $\wedge$ `(= (+ b c) d)`) $\Rightarrow$ `(= a (+ b c))` is an instance of (`(= x y)` $\wedge$ `(= y z)`) $\Rightarrow$ `(= x y)`, where `x` has been replaced by `a`, `y` has been replaced by `(+ b c)` and `z` has been replaced by `d`.

An *axiom* is a formula that is given to be valid, without proof. In a precise sense, axioms correspond to the primitives of the ACL2 language. This is the logic equivalent of having a primitive in the language, for which you cannot see the definition in terms of other primitives. An axiom we shall see, for example, is that `(= (car (cons x y)) x)`.

So what's an equational proof?

Definition. *An* equational proof *of a formula F of ACL2 logic is a sequence of formulas $F_1, F_2, \ldots, F_k$ such that:*

- *the first formula F_1 is just F;*
- *the last formula F_k is either an axiom or an instance of a* valid *propositional formula*

- *every formula in the sequence is obtained from the preceeding one by one of the* Equational Proof Rules *detailed below.*

We will consider, at first, three equational proof rules that rewrite formulas into other formulas.

Rule of Substitution: If $G \equiv G'$ is an instance of a theorem, then we can rewrite a formula F by replacing occurrences of G in F by G'.

Rule of Propositional Reasoning: If $\varphi \equiv \psi$ is a valid propositional formula, and $G \equiv G'$ is an instance of $\varphi \equiv \psi$, then we can rewrite a formula F by replacing occurrences of G in F by G'.

Rule of Modus Ponens: if $G \Rightarrow G'$ is an instance of a theorem, then we can rewrite a formula F by replacing occurrences of G in F by $G \wedge G'$.

Those are the basic rules. Note that Propositional Reasoning lets us do a vast amount of propositional reasoning in one shot. (You may need to have an argument on the side showing that a particular propositional formula you need is valid.)

It is absolutely immediate that every axiom is a theorem, and similalry that every instance of a valid propositional formulas is also a theorem.

The reason why we care about equational proofs is the following **Soundness Property of ACL2 Logic**: *If an ACL2 formula F has an equational proof, that is, if F is a theorem, then F is a valid formula of ACL2 logic.*

In other words, if F has an equational proof, then we can be sure that for any values of the variables in F, F will be true. This result takes the place of potentially an infinite number of tests that you would need to perform.[1] The reason why this Soundness Property is true is that a proof is a sequence of formulas in which each formula is valid if and only if the previous formula is valid. By arriving at a formula that is valid at the end (an axiom or an instance of a valid propositional formula), this means that the whole sequence of formulas represents valid formulas, so the first one must be valid as well.

In order to get more than just propositional validities, we need to know something about the expressions in ACL2. After all, primitive operations such as `=` and `car` and the likes have some properties that we haven't captured yet. Those are the axioms I've talked about above. So what are the axioms? For the time being, that is, until next lecture, let's focus on formulas in which the ACL2 expressions are of the form `(= exp1 exp2)` where `exp1` and

[1] The Soundness Property is a mathematical property of the logic. Because it is a mathematically statement, it can actually be proved, and the proof is actually not that hard, although it is somewhat outside the scope of this part of the course. If you're curious, and you have the mathematical background, the proof is a simple proof by induction on the length of an equational proof—you only need to show that every rule preserves and reflects the validity of the formulas involved.

`exp2` are arbitrary expressions. The axioms controlling `=` and function symbols are simple, and we've seen most of them last time.

Axioms for Equality:

- reflexivity: `(= x x)`
- symmetry: `(= x y)` $\equiv$ `(= y x)`[2]
- transitivity: `(= x y)` $\wedge$ `(= y z)` $\equiv$ `(= x y)` $\wedge$ `(= x z)`[3]
- Leibniz: (`(= x1 y1)` $\wedge \ldots \wedge$ `(= xn yn)`) $\Rightarrow$ `(= (f x1 ... xn) (f y1 ... yn))` for any symbol `f` of arity n

Note that the last axiom is really a family of axioms, one per symbol `f`).

Here is an example.

Theorem. `(= x y)` $\wedge$ `(= (foo y) (bar y))` $\Rightarrow$ `(= (foo x) (bar y))`

Proof.

`(= x y)` $\wedge$ `(= (foo y) (bar y))` $\Rightarrow$ `(= (foo x) (bar y))`

by Modus Ponens with Leibniz axiom

`(= x y)` $\wedge$ `(= (foo x) (foo y))` $\wedge$ `(= (foo y) (bar y))` $\Rightarrow$ `(= (foo x) (bar y))`

by Substitution with transitivity of `=`

`(= x y)` $\wedge$ `(= (foo x) (foo y))` $\wedge$ `(= (foo x) (bar y))` $\Rightarrow$ `(= (foo x) (bar y))`

which is an instance of the valid $(p \wedge q) \Rightarrow q$

$\square$

Next time, we look at more general expressions in ACL2 formulas, and how to reason about functions we define ourselves.

[2]Equivalently, we could use `(= x y)` $\Rightarrow$ `(= y x)`—we could prove the same theorems.

[3]Equivalently, we could use `(= x y)` $\wedge$ `(= y z)` $\Rightarrow$ `(= x z)`—we could prove the same theorems.