

Announcements

- Exam 2 is tomorrow.
- Reading for this part of course: 6.1-6.2,6.4

Last time: conjectures and theorem proving

Program testing can be used to show the presence of bugs, but never to show their absence! EWD249, 1970

Today: The ACL2 Logic

First, we start with propositional reasoning:

We are going to assume that every propositional tautology is an axiom.

What is a propositional tautology?

```
a /\ ~a
a => b iff ~a /\ b
(a /\ b) => c iff a => (b => c)
```

In ACL2:

```
(thm (or a (not a)))
(thm (iff (implies a b) (or (not a) b)))
(thm (iff (implies (and a b) c)
          (implies a (implies b c))))
```

What about:

```
a /\ b <-> a /\ b <-> a <-> b
(thm (iff (iff (and a b) (or a b)) (iff a b)))
```

So, I am going to assume that we can do propositional reasoning correctly, but let's quickly review.

Construct truth table for and, or, not, =>, iff.

Construct the truth table for ite.

How do we define and, or, and not in terms of ite (we have true and false)?

Question: how many Boolean functions over 2 variables are there? 16 (how many 4 bit vectors?)

A tautology can be recognized by using a truth table to test the value of the formula on each of a finite number of assignments. Rather than exhibit the formal proof, we can just exhibit the truth table and appeal to this well-known result.

Checking if a formula is a tautology (satisfiable) is a *key* problem in computer science. For example whether it can be done in polynomial time is the most important theoretical question in CS and one of the most important problems in mathematics.

 Aside

In the book, we have the following:

Propositional Axiom Schema: $(\sim\phi \vee \phi)$

This means that we have such an axiom for every formula ϕ .

We also have four rules of inference:

Expansion: derive $(\phi_1 \vee \phi_2)$ from ϕ_2 ;

Contraction: derive ϕ from $\phi \vee \phi$;

Associativity: derive $((\phi_1 \vee \phi_2) \vee \phi_3)$ from $(\phi_1 \vee (\phi_2 \vee \phi_3))$; and

Cut: derive $(\phi_2 \vee \phi_3)$ from $(\phi_1 \vee \phi_2)$ and $(\neg \phi_1 \vee \phi_3)$.

Here is an amazing result: every propositional tautology has a proof with the above axioms and rules of inference.

We are not going to show this; we are just going to use it.

 End Aside

When you are defining a logic, you have to give clear rules as to what constitutes a proof.

A formal proof is a finite sequence of formulas, each of which is either an axiom or is derived from previous formulas by one of the rules of inference above.

A theorem is any formula in a proof, but most especially the last formula.

So, you are free to write down the following proof:

$(\text{iff} (\text{iff} (\text{and } a \ b) \ (\text{or } a \ b)) \ (\text{iff } a \ b))$: propositional tautology

Since all propositional tautologies are axioms.

Let's look more carefully at Boolean connectives in ACL2.

They are all defined in terms of `if` (which is similar to `ite`). Here is the exact semantics of `if`:

$x = \text{nil} \Rightarrow (\text{if } x \ y \ z) = z$
 $x \neq \text{nil} \Rightarrow (\text{if } x \ y \ z) = y$

Note that `if` is `ite` on boolean arguments, but `if` is really total. So, $(\text{if } 3 \ 4 \ 5) = ??$

How are the rest of the Boolean connectives defined?

$(\text{and } x \ y) = (\text{if } x \ y \ \text{nil})$
 $(\text{or } x \ y) = (\text{if } x \ x \ y)$
 $(\text{not } x) = (\text{if } x \ \text{nil} \ t)$
 $(\text{iff } x \ y) = (\text{if } x \ (\text{if } y \ t \ \text{nil}) \ (\text{if } y \ \text{nil} \ t))$
 $(\text{implies } x \ y) = (\text{if } x \ (\text{if } y \ t \ \text{nil}) \ t)$

We also have certain axioms about equality.

Reflexivity: $x = x$

Equality Axiom Schema for Functions: For every function symbol of arity n , we have:

Equality Axiom Schema for Functions:
For every function symbol f of arity n we add the axiom

$$x_1 = y_1 \wedge \dots \wedge x_n = y_n \Rightarrow (f x_1 \dots x_n) = (f y_1 \dots y_n)$$

Equality Axiom:

$$x = y \wedge w = z \wedge x = w \Rightarrow y = z$$

Obviously, `equal`, is also important, and here is what we know about `equal`.

$$x = y \Rightarrow (\text{equal } x \ y) = t$$

$$x \neq y \Rightarrow (\text{equal } x \ y) = \text{nil}$$

Another axiom relates `car`, `cons`, and `equal`.

$$(\text{equal } (\text{car } (\text{cons } x \ y)) \ x) \neq \text{nil}$$

More ACL2 axioms:

$$(\text{cdr } (\text{cons } x \ y)) = y$$

$$(\text{consp } (\text{cons } x \ y)) = t$$

We can similarly axiomatize four other data types: the complex rationals (with the rationals and integers as subsets), the characters, the strings, and the symbols (with packages). These data types are pairwise disjoint. For example we have

$$(\text{implies } (\text{consp } x) (\text{not } (\text{symbolp } x)))$$

which together with
`(symbolp nil)`

implies that `(consp nil) = nil`.

The axioms for arithmetic are essentially the "usual" ordered field axioms for constants 0 and 1 and function symbols `+`, `-`, `*`, `/`, and `<`. Since the language is not typed, terms such as `(+ nil 1)` are legal. Our axioms "complete" the primitive functions. For example, when an argument to `+` is not numeric, that argument is treated as though it were 0.