Theorems and Theories

Is the following a theorem?

```
(implies (> x 20000)
         (too-big x))
```

Well, if we have not defined the function `too-big`, then it certainly is not a theorem and ACL2 won't even attempt to prove the proposition.

If we make this definition,

```
(defun too-big (x)
  (> x 1000))
```

then the theorem is clearly true. ACL2 proves it:

```
(thm (implies (> x 20000)
              (too-big x)))
```

But if we define `too-big` differently,

```
(defun too-big (x)
  (> x 1000000))
```

it is not a theorem. Here's a counterexample:

```
(check=
 (let ((x 30000))
   (implies (> x 20000)
            (too-big x)))
 nil)
```

The point here is that whether a proposition is a theorem or not depend on the current *theory*, which depends on all the function definitions.

Theories and Axioms

The definitional principle says that a definition like

```
(defun too-big (x)
  (> x 1000))
```

adds a new axiom to the current logical theory:

```
(equal (too-big x) (> x 1000))
```

What does this mean? In fact, a logical **theory** is a set of axioms, and an **axiom** is a proposition that is *assumed* to be true. An axiom is a fundamental truth about the system we are working on.

What do axioms do for us? That is where a **logic** comes in, with **rules of inference**, which allow us to derive theorems from axioms and other theorems.

This is the alternate characterization of theorems, instead of saying a theorem is a valid(*true* in all possible assignments to free variables) formula, we can define:

Thm := Axiom | application of finite number of Rules of Inference to Thm

Its amazing that one can decide if a statement is a theorem in this manner, instead of the standard way of checking all possibilities. This is how theorem provers work.

Now we have a picture of where proofs of theorems come from: a theorem concerns a given theory in a given logic. That theory is a set of axioms. The logic has rules of inference that allow us to generate other theorems from those axioms. (Axioms are theorems.)

When we start ACL2, it has lots of functions already defined and it correspondingly has axioms for those functions in its theory. Remember from the Lecture 5 that some functions are "primitive" and some are "derived?" The axioms governing derived functions come from the definitional principle. For example, there is an axiom that says

```
(equal (true-listp x)
       (if (consp x)
         (true-listp (cdr x))
         (equal x nil)))
```

which comes from the definition of true-listp.

Primitive functions, however, are not defined in terms of other functions, so the axioms governing those were put together carefully and cleverly by the people who created ACL2. Here are some examples of those:

```
(equal x x)
(not (equal t nil))
(implies (equal x nil)
         (equal (if x y z)
                z))
(implies (not (equal x nil))
         (equal (if x y z)
                y))
```

Rules of inference

The textbook describes a small, rather cryptic set of inference rules for ACL2. There is an elegance in making a logic as simple as possible, but it can be hard to work with. I will describe a more general set of inference rules that should be easy to understand and more easily usable. Here are some:

Instantiation: If φ is a theorem, then φ with all occurences of a free variable replaced by some expression is also a theorem. We could also state this as

$\varphi \Rightarrow$ `(let ((`v e`)) φ)`

because the meaning of LET is to replace free occurrences of v with e. However, we don't want to depend on LET for this rule, so go ahead and do the replacement.

For example,

```
(consp (cons a b))
```

is an axiom and, therefore, a theorem. We can **instantiate** `a` with any expression, including `(car x)`, and get another theorem:

```
(consp (cons (car x) b))
```

The rule of inference tells us this must be a theorem. Furthermore, we can instantiate `b` to get yet another theorem:

```
(consp (cons (car x) (app (cdr x) y)))
```

Propositional deduction: If

$(\varphi_1 \bigwedge \varphi_2 \bigwedge \ldots \bigwedge \varphi_n) \rightarrow \psi$

is a propositional tautology—a tautology according to boolean logic—and φ_1, φ_2, ..., φ_n are theorems, then we can conclude ψ is a theorem.

Note: Since any proposition tautology(or a number of them) can be written in the above form, we can be more general and say, “Any finite number of proposition tautologies(boolean equalites) seen in class, can be used to make a propositional deduction”. This is very intuitive since, proposition logic is just a small subset of ACL2 logic and so any “reasoning” we used in the simplification proofs seen earlier in the class can be applied to ACL2 formulas(propositions).

In other words,

$\varphi_1, \varphi_2, \ldots, \varphi_n \Rightarrow \psi$ if $(\varphi_1 \bigwedge \varphi_2 \bigwedge \ldots \bigwedge \varphi_n) \rightarrow \psi$ is a tautology

For example, from these ACL2 axioms,

```
(implies (integerp x) (acl2-numberp x))
```

```
(implies (consp x) (not (acl2-numberp x)))
```

I claim I can deduce

```
(implies (integerp x) (not (consp x)))
```

I can get this by a propositional deduction. If you let i stand for `(integerp x)`, n for `(acl2-numberp x)` and c for `(consp x)`, the following must be a tautology for the deduction to be legal:

$((i \rightarrow n) \bigwedge (c \rightarrow \neg n)) \rightarrow (i \rightarrow \neg c))$

Well, $c \rightarrow \neg n$ is equivalent to $n \rightarrow \neg c$, and $(i \rightarrow n) \bigwedge (n \rightarrow \neg c)$ implies $i \rightarrow \neg c$. Thus, it is a tautology. (Check it with a truth table.) Thus, the deduction above is a

legal application of propositional deduction. Thus, `(implies (integerp x) (not (consp x)))` is a theorem.

A special case of propositional deduction is **Modus ponens**. It says that if we have theorems φ and (`implies` φ ψ), then we can conclude ψ. That is because (φ ⋀ (φ → ψ)) → ψ is a tautology.

Equals for equals: If we have a theorem that e_1 is equal to e_2 and another theorem φ, then we can conclude φ with occurrences of e_1 in it optionally replaced by e_2.

For example, if we have theorems

```
(equal (> x 1000) (too-big x))
```

```
(implies (> x 20000)
         (> x 1000))
```

Then we can **replace equals for equals** to conclude

```
(implies (> x 20000)
         (too-big x))
```

In this case, e_1 would be `(> x 1000)`, e_2 would be `(too-big x)`, and φ would be `(implies (> x 20000) (> x 1000))`.

Sample Proof

I will prove

```
(not (consp (len x)))
```

I will assume a theorem I proved above,

```
(implies (integerp x) (not (consp x)))
```

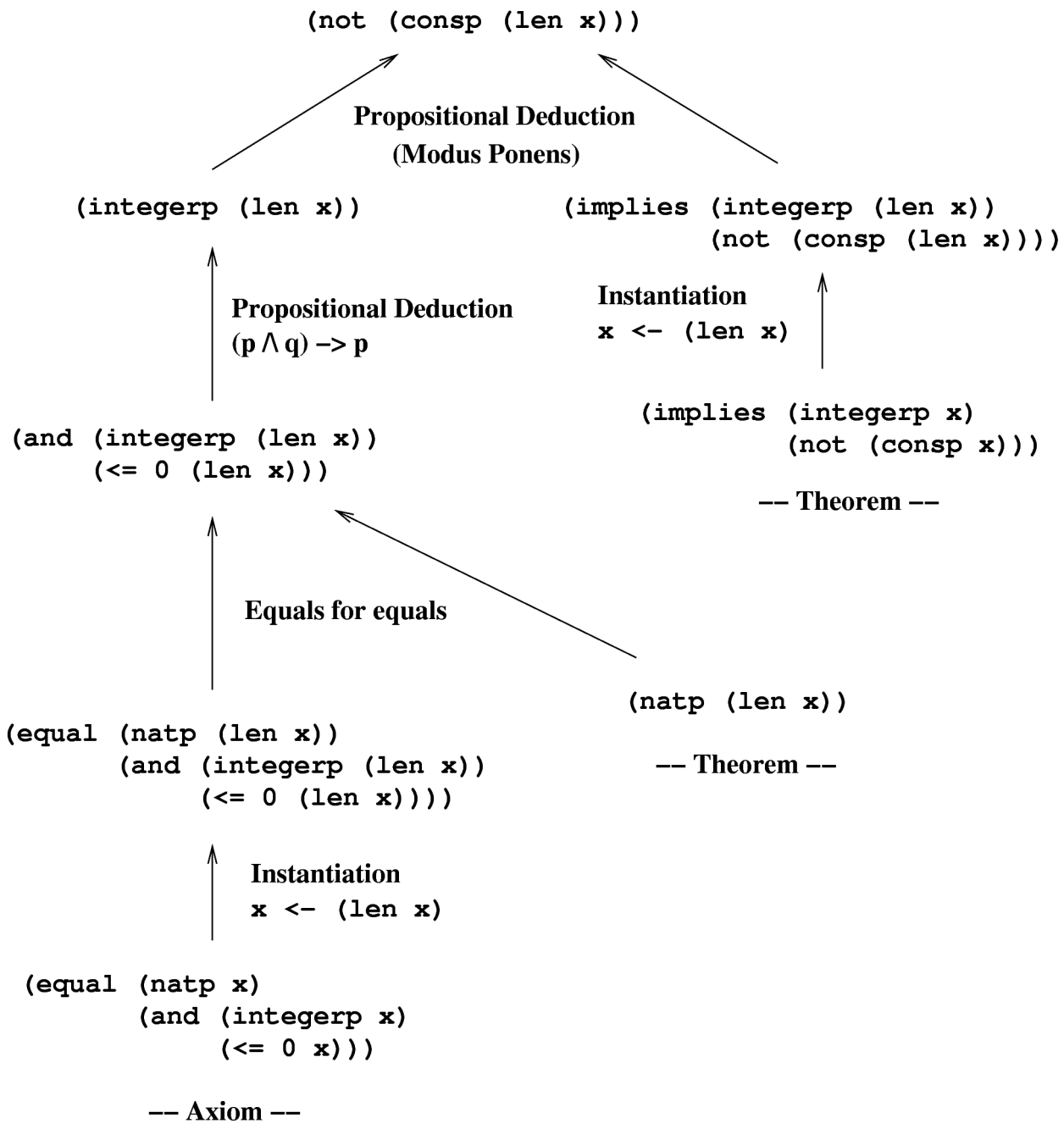
another I will not prove here,

```
(natp (len x))
```

and the definitional axiom for NATP:

```
(equal (natp x)
       (and (integerp x) (<= 0 x)))
```

Here is a pictorial proof, that starts from the axioms or other known theorems and applies Rules of Inference to eventually obtain the formula we need to prove. And hence by definition, the above is a theorem. I show this proof format only to explain the formal definition of a proof, but its sometimes hard to do it in this fashion and in the future we will stick to the format we used in Simplification, we start from the formula we want to prove and transform it into either *true* or a known axiom/theorem, which completes the proof.


(not (consp (len x)))
Propositional Deduction
(Modus Ponens)
(integerp (len x))
(implies (integerp (len x))
(not (consp (len x))))
Propositional Deduction
(p ∧ q) -> p
Instantiation
x <- (len x)
(and (integerp (len x))
(<= 0 (len x)))
(implies (integerp x)
(not (consp x)))
-- Theorem --
Equals for equals
(natp (len x))
-- Theorem --
(equal (natp (len x))
(and (integerp (len x))
(<= 0 (len x))))
Instantiation
x <- (len x)
(equal (natp x)
(and (integerp x)
(<= 0 x)))
-- Axiom --

A **formal proof** is a tree of propositions whose tips (“leaves”) are axioms and each other node is a consequence of its children by some rule of inference. The root node has the overall theorem that is proven.

Technically, the proof above is not a complete formal proof, because some of the tips are just known theorems rather than axioms. For it to be a complete formal proof, one would need to fill in proofs of those theorems, but it is standard to use previously proven theorems in subsequent proofs.

Note: Lecture notes were derived from notes by Peter Dillinger.