Logic and Computation – CS 2800 Fall 2019

Lecture 14

Equational reasoning

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

A couple more notes on Boolean logic

- Boolean operators have a correspondence with classic operators on sets:
- Let $[\![\phi]\!]$ be the set of assignments that make ϕ true
- Conjunction = intersection
  - $[\![\phi_1 \wedge \phi_2]\!] = [\![\phi_1]\!] \cap [\![\phi_2]\!]$
- Disjunction = union
  - $[\![\phi_1 \vee \phi_2]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!]$
- Negation = complementation
  - $[\![\neg\phi]\!] = \overline{[\![\phi]\!]}$
- Implication = subset relation
  - The formula $\phi_1 \rightarrow \phi_2$ is valid iff $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$

A couple more notes on Boolean logic

- Implication = subset relation
  - The formula $\phi_1 \rightarrow \phi_2$ is valid iff $[\![\phi_1]\!] \subseteq [\![\phi_2]\!]$
- Then it's easy to see why $\phi_1 \vee (\phi_1 \wedge \phi_2) \equiv \phi_1$:
  - $(\phi_1 \wedge \phi_2) \rightarrow \phi_1$, so $[\![\phi_1 \wedge \phi_2]\!] \subseteq [\![\phi_1]\!]$
  - $[\![\phi_1 \vee (\phi_1 \wedge \phi_2)]\!] = [\![\phi_1]\!] \cup [\![\phi_1 \wedge \phi_2]\!] = [\![\phi_1]\!]$

Quiz

- Does XOR distribute over disjunction?
- i.e., is this equivalence valid?
  - $a \oplus (b \vee c) \equiv (a \oplus b) \vee (a \oplus c)$

A. Yes
B. No

Outline

- Beyond propositional logic
- Equational reasoning: an introduction

Beyond Boolean logic

Non-Boolean formulas

- Consider the formula: $x + y = y + x$
- Suppose that variables x and y are rational numbers, and + is the usual arithmetic addition
- Then, this is not a propositional logic formula
- But it's still a valid formula: for any pair of x and y, the formula evaluates to true
- How to prove that?
- We can't construct the entire truth table: it's infinite!

Finite work, infinite results

- We need a **finite proof**
- The proof will be finite, but it will imply an “infinite result”, namely, that the formula is true **for all rational numbers x and y**

- **Equational reasoning**: a class of such finite proofs

- We are now getting into the heart of proving program correctness!

Example

```
(definec len (l :all) :nat
  (if (consp l)
      (+ 1 (len (cdr l)))
      0))
```

- Conjecture 1:
  - `(equal (len (list x)) (len x))`
  - True or false?
  - False: e.g., let x=1
  -
```
(equal (len (list 1)) (len 1))
    = (equal (+ 1 (len nil)) 0)
    = (equal (+ 1 0) 0)
    = (equal 1 0)
    = nil
```
- Checking a candidate counterexample is easy: just plug in the values and evaluate

Example

```
(definec len (l :all) :nat
  (if (consp l)
      (+ 1 (len (cdr l)))
      0))
```

- Conjecture 2:
  - `(thm (equal (len (cons x z))`
    `(len (cons y z))))`
  - True or false?
  - True!
  - We need a finite proof…

Equational proof

```
(definec len (l :all) :nat
  (if (consp l)
    (+ 1 (len (cdr l)))
    0))
```

Goal: (len (cons x z)) = (len (cons y z))

```
(len (cons x (list z)))
= {                                          }
… continue …
```

Equational proof

```
(definec len (l :all) :nat
  (if (consp l)
    (+ 1 (len (cdr l)))
    0))
```

```
(len (cons x z))
= { definition of len }
(if (consp (cons x z))
    (+ 1 (len (cdr (cons x (list z)))))
    0))
= { consp axioms }
(if t
    (+ 1 (len (cdr (cons x z))))
    0))
= { if axioms }
    (+ 1 (len (cdr (cons x z))))
= { cdr axioms }
    (+ 1 (len z))
```

Equational proof

```
(definec len (l :all) :nat
  (if (consp l)
    (+ 1 (len (cdr l)))
    0))
```

- So far, we proved this:

```
(len (cons x z))
=
(+ 1 (len z))
```

- But we wanted this:

```
(equal (len (cons x z))
       (len (cons y z)))
```

- How to continue?

Equational proof

```
(definec len (l :all) :nat
  (if (consp l)
    (+ 1 (len (cdr l)))
    0))
```

- We have this:

```
(len (cons x z))
= (+ 1 (len z))
```

- We want this:

```
(len (cons x z))
=  (len (cons y z)))
```

- We have several options:
  - We can start from the right-hand side and prove

```
(len (cons y z)) = (+ 1 (len z))
```

  - We can use a **lemma**

Lemmas

```
(definec len (l :all) :nat
  (if (consp l)
    (+ 1 (len (cdr l)))
    0))
```

- Prove them once, use them again and again!
- “helper theorems” (like helper functions)
- Our Lemma:

```
(len (cons x z)) = (+ 1 (len z))
```

- The Lemma holds **for any x and z**!
- So we can instantiate it with y instead of x:

```
(len (cons y z)) = (+ 1 (len z))
```

- Instantiation is like “calling” the lemma with different arguments (like calling a helper function)

Proof using the Lemma

```
(definec len (l :all) :nat
  (if (consp l)
    (+ 1 (len (cdr l)))
    0))
```

```
(len (cons x z))
= { Lemma }
(+ 1 (len z))
= { Lemma with instantiation ((x y)) }
(len (cons y z))
```

Note: here we implicitly used symmetry of equality:
If A = B then B = A

```
Lemma:
(len (cons x z)) = (+ 1 (len z))
```

Recap: proofs

- Finite proofs: finite number of steps
- Justification for each step
- Beyond propositional logic:
  - non-Boolean variables, ACL2s functions, axioms of predefined functions, equality properties, lemmas, instantiations, …
- In the end, we can prove a very strong result:
  - That something holds for any object in the ACL2s universe!

Equality

Equality

- Equality is an **equivalence relation**
  - Reflexive: for all $x : x = x$
  - Symmetric: for all $x, y : x = y \Rightarrow y = x$
  - Transitive: for all $x, y, z : (x = y \wedge y = z) \Rightarrow x = z$
- The above properties are **axioms**:
  - Things we take for granted (we don't have to prove them)
  - We can use any axiom in our proofs!
  - We use axioms like we use lemmas: we can instantiate them with anything we like, e.g., `42=42`, `nil=nil`, `"hello"="hello"`, `...`
- In ACL2s, we write `(equal x y)`
  - On paper, slides, whiteboard, etc., we can also write `x = y`

Equality

- **Equality Axiom Schema for Functions**:
- For every function symbol f of arity n, we have the axiom:
  - For all $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n$:

$$(x_1 = y_1 \wedge x_2 = y_2 \wedge \cdots \wedge x_n = y_n) \Rightarrow$$
$$(f\ x_1\ x_2\ \ldots\ x_n) = (f\ y_1\ y_2\ \ldots\ y_n)$$

- Example:

```
(implies (equal x y) (equal (len x) (len y)))
```

Axioms for built-in functions

Some axioms for built-in functions

(from now on we drop the "for all x, y, …" part, which is implicit)

- Axioms for `if`:
  - `x = nil => (if x y z) = z`
  - `x ≠ nil => (if x y z) = y`
- Axioms for `car, cdr`:
  - `(car (cons x y)) = x`
  - `(cdr (cons x y)) = y`
- Axiom for `consp`:
  - `(consp (cons x y)) = t`

Next time

- Equational reasoning continued