Logic and Computation – CS 2800 Fall 2019

Lecture 16

Equational reasoning continued

Exportation

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

Some notes about homework 05

- (x . nil) vs (x)
- assignment and alistof
- all-assignments

- Go to the labs!

Outline

- Equational reasoning continued
- More examples
- The propositional skeleton
- Exportation

Quiz : all answers graded 8 points, no answer = 0 points

A. I have done the proof of example 3 and I am willing to present it on the whiteboard
B. I have done the proof of example 3 but I am not willing to present it on the whiteboard
C. I have not done the proof of example 3

Example 3

```
(definec aapp (x :tl y :tl) :tl
  (if (endp x)
      y
    (cons (car x) (aapp (cdr x) y))))
```

- The same claim but with renamed variables:

```
((tlp a) & (tlp b) & (tlp c) & (consp a) &
(aapp (aapp (cdr a) b) c) = (aapp (cdr a) (aapp b c)))
=> (aapp (aapp a b) c) = (aapp a (aapp b c))
```

```
Endp-consp axioms:
(tlp x) => ((endp x) =
                (not (consp x)))
```

```
(definec aapp (x :tl y :tl) :tl
  (if (endp x)
      y
    (cons (car x) (aapp (cdr x) y))))
```

- Example 3 detailed proof:

```
Context:
C1. (tlp a)
C2. (tlp b)
C3. (tlp c)
C4. (consp a)
C5. (aapp (aapp (cdr a) b) c) = (aapp (cdr a) (aapp b c))
Derived context:
D1. (not (endp a)) { C1, C4, endp-consp axioms, Modus Ponens }
Goal:  (aapp (aapp a b) c) = (aapp a (aapp y z))
Proof:
(aapp (aapp a b) c)
= { def aapp }
(aapp (if (endp a)
          b
          (cons (car a)
                (aapp (cdr a) b))) c)
= { D1 }
(aapp (if nil
          b
          (cons (car a)
                (aapp (cdr a) b))) c)
= { if axioms }
(aapp (cons (car a) (aapp (cdr a) b)) c)
= { def aapp }
(if (endp (cons (car a) (aapp (cdr a) b)))
    c
    (cons (car (cons (car a) (aapp (cdr a) b)))
          (aapp (cdr (cons (car a) (aapp (cdr a) b)))
                c)))
= { consp axioms }
(if nil
    c
    (cons (car (cons (car a) (aapp (cdr a) b)))
          (aapp (cdr (cons (car a) (aapp (cdr a) b)))
                c)))
= { if axioms }
    (cons (car (cons (car a) (aapp (cdr a) b)))
          (aapp (cdr (cons (car a) (aapp (cdr a) b)))
                c))
= { car-cdr axioms (twice) }
    (cons (car a)
          (aapp (aapp (cdr a) b)
                c))
= { get rid of 2nd line break }
    (cons (car a)
          (aapp (aapp (cdr a) b) c))
= { C5 }
    (cons (car a)
          (aapp (cdr a) (aapp b c)))
= { def aapp, D1 }
(aapp a (aapp b c))
```

```
Endp-consp axioms:
(tlp x) => ((endp x) =
                (not (consp x)))
```

```
(definec aapp (x :tl y :tl) :tl
  (if (endp x)
      y
    (cons (car x) (aapp (cdr x) y))))
```

- Example 3 proof split in multiple parts:

```
Context:
C1. (tlp a)
C2. (tlp b)
C3. (tlp c)
C4. (consp a)
C5. (aapp (aapp (cdr a) b) c) = (aapp (cdr a) (aapp b c))
Derived context:
D1. (not (endp a)) { C1, C4, endp-consp axioms, Modus Ponens }
Goal:  (aapp (aapp a b) c) = (aapp a (aapp y z))
Proof:
```

Example 3 proof split in multiple parts, continued:

```
Proof:
(aapp (aapp a b) c)
= { def aapp }
(aapp (if (endp a)
          b
          (cons (car a)
                (aapp (cdr a) b))) c)
= { D1 }
(aapp (if nil
          b
          (cons (car a)
                (aapp (cdr a) b))) c)
= { if axioms }
(aapp (cons (car a) (aapp (cdr a) b)) c)
= { def aapp }
(if (endp (cons (car a) (aapp (cdr a) b)))
    c
    (cons (car (cons (car a) (aapp (cdr a) b)))
          (aapp (cdr (cons (car a) (aapp (cdr a) b)))
                c)))
= { consp axioms }
(if nil
    c
    (cons (car (cons (car a) (aapp (cdr a) b)))
          (aapp (cdr (cons (car a) (aapp (cdr a) b)))
                c)))
= { if axioms }
    (cons (car (cons (car a) (aapp (cdr a) b)))
          (aapp (cdr (cons (car a) (aapp (cdr a) b)))
                c))
= { car-cdr axioms (twice) }
    (cons (car a)
          (aapp (aapp (cdr a) b)
                c))
= { get rid of 2nd line break }
    (cons (car a)
          (aapp (aapp (cdr a) b) c))
= { C5 }
    (cons (car a)
          (aapp (cdr a) (aapp b c)))
= { def aapp, D1 }
(aapp a (aapp b c))
```

The propositional skeleton

Example 4

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))
```

- Claim:

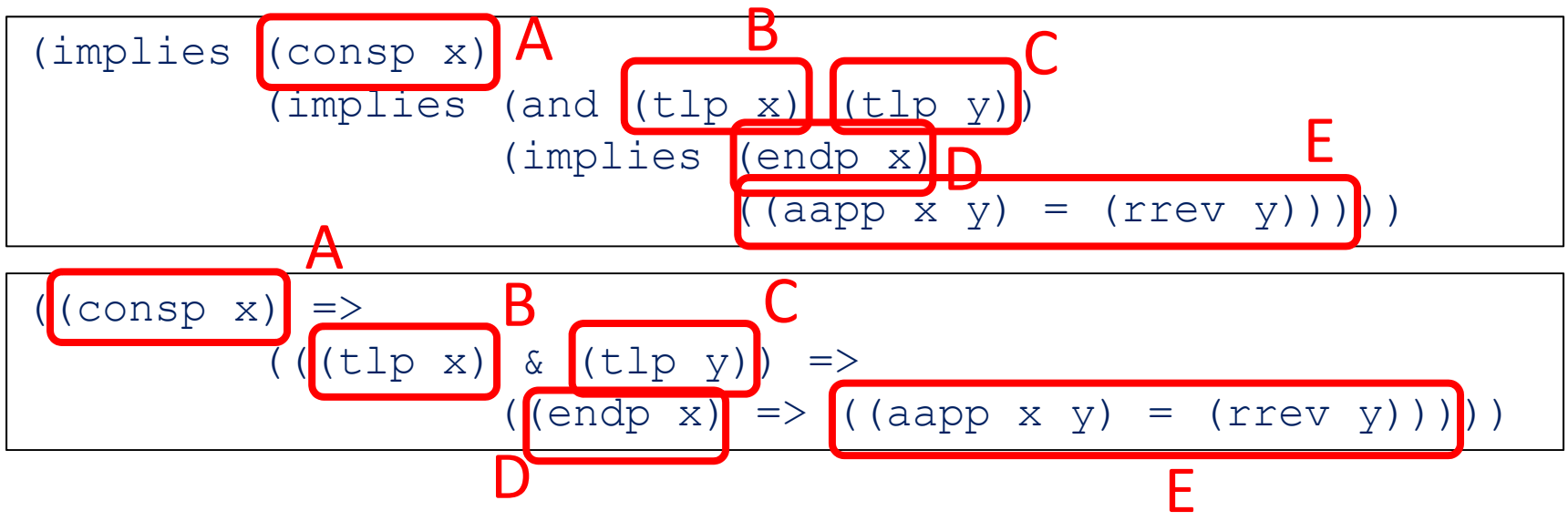
```
(implies (consp x)
         (implies (and (tlp x) (tlp y))
                  (implies (endp x)
                           (equal (aapp x y) (rrev y)))))
```

- Let's first make this more humanly readable:

```
((consp x) =>
         (((tlp x) & (tlp y)) =>
                  ((endp x) => ((aapp x y) = (rrev y)))))
```

The propositional skeleton

- The propositional structure of the formula:




```
( A =>
        (( B & C ) =>
                ( D => E )))
```

The propositional skeleton

Exportation

Exportation

- Separate the hypotheses from the proof goal
- Using properties of propositional logic
- During exportation, we only work with the propositional skeleton:

```
( A => (( B & C ) => ( D => E )))
=
( A & B & C & D ) => E
```

- Why does this work?
- Propositional logic exportation rule:
- (and also associativity of conjunction)

$$p \rightarrow (q \rightarrow r)$$
$$\equiv$$
$$(p \wedge q) \rightarrow r$$

Exportation: be careful

- What does exportation give on this?

```
( A => (( B => C ) => ( D => E )))
                =
              ???
```

Exportation: be careful

- What does exportation give on this?

$$(A \Rightarrow ((B \Rightarrow C) \Rightarrow (D \Rightarrow E)))$$
$$=$$
$$(A \,\&\, (B \Rightarrow C) \,\&\, D) \Rightarrow E$$

CORRECT!

- Many students make this mistake:

$$(A \Rightarrow ((B \Rightarrow C) \Rightarrow (D \Rightarrow E)))$$
$$=$$
$$(A \,\&\, B \,\&\, C \,\&\, D) \Rightarrow E$$

WRONG!

Exportation: be careful

- Why is this wrong?

```
( A => (( B => C ) => ( D => E )))
                =
      ( A & B & C & D ) => E
```

$$p \rightarrow (q \rightarrow r)$$
$$=$$
$$(p \wedge q) \rightarrow r$$

$$(p \rightarrow q) \rightarrow r$$
$$\neq$$
$$(p \wedge q) \rightarrow r$$

Back to our Example 4

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))
```

- Claim:

```
(implies (consp x)
         (implies (and (tlp x) (tlp y))
                  (implies (endp x)
                           (equal (aapp x y) (rrev y)))))
```

- Is this claim true?

```
((consp x) =>
         (((tlp x) & (tlp y)) =>
                  ((endp x) => ((aapp x y) = (rrev y)))))
```

Back to our Example 4

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))
```

- Is this claim true?

```
((consp x) =>
         (((tlp x) & (tlp y)) =>
                  ((endp x) => (aapp x y) = (rrev y))))
```

- Yes!
- Exportation gives:

```
((consp x) & (tlp x) & (tlp y) & (endp x))
              => (aapp x y) = (rrev y)
```

- The hypotheses are contradictory!

Example 4

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))
```

```
consp-endp axioms:
        (tlp x) & (endp x) => (not (consp x))
        (tlp x) & (consp x) => (not (endp x))
```

- Proof:

```
Context:
C1: (consp x)
C2: (tlp x)
C3: (tlp y)
C4: (endp x)

Derived context:
D1: nil { C1, C2, C4, consp-endp axioms }
```

QED!

Once you derive nil (false) as a hypotheses, any goal is proven!

Next time

- Equational reasoning continued