Logic and Computation – CS 2800 Fall 2019

Lecture 17

Equational reasoning continued

Exportation, more examples

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

Outline

- Equational reasoning continued
- The automated proof checker
- Exportation
- More examples

Automated proof checker

- http://checker.atwalter.com/
  - Thanks to Andrew Walter for this tool

- Let's run example 3 on the proof checker!

- For homeworks:
  - You are **not required** to use the proof checker
  - The proof checker is difficult to use:
    - Error reporting sucks
    - E.g., "Derived context:" vs. "Derived Context:"
    - Another example: what's wrong with my Example 3 slides?
  - We can only provide minimal support with the proof checker
  - Your proof passing the proof checker, **is neither a necessary nor a sufficient condition for it to be an acceptable proof**

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

CORRECT!

- Many students make this mistake:

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

WRONG!

Exportation: be careful

- Why are these different?

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

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

$$(p \to q) \to r \not\equiv (p \wedge q) \to r$$

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!

Quiz

- Which of the formulas below are valid?
  1. $\big(A \to (B \to C)\big) \quad \equiv \quad \big((A \wedge B) \to C\big)$
  2. $(A \to B) \quad \equiv \quad (\neg B \to \neg A)$
  3. $((A \to B) \wedge A) \ \to B$
  4. $\big((A \to B) \wedge \neg B\big) \to A$

A. Only (1)
B. (1), (2) and (3)
C. (1), (2) and (4)
D. All four

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Claim:

```
(((tlp x) & (tlp y)) =>
      ((endp x) =>
          ((not (in a (aapp x y))) =>
              (not (in a x)))))
```

- First step?

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Claim:

```
(((tlp x) & (tlp y)) =>
      ((endp x) =>
          ((not (in a (aapp x y))) =>
              (not (in a x)))))
```

- Propositional skeleton:

```
((A & B) =>
      (C =>
          ((not D) => (not E))))
```

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Propositional skeleton:

```
((A & B) =>
      (C =>
          ((not D) => (not E))))
```

- Contraposition:

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

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- After contraposition:

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

- Next step?

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- After contraposition:

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

- Next step?
- **Exportation!**

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

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Original claim:

```
(((tlp x) & (tlp y)) =>
      ((endp x) =>
          ((not (in a (aapp x y))) =>
              (not (in a x)))))
```

- New (equivalent) claim:

```
((tlp x) & (tlp y) & (endp x) & (in a x)) =>
                  (in a (aapp x y))
```

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Next step: **proof!**

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

Goal: (in a (aapp x y))

Proof:
???
```

Example 5

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Next step: **proof!**

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

Derived context:
D1: nil { C1, C3, C4, def in }

Goal: (in a (aapp x y))
```

QED!

Example 6

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- Claim:

```
(implies (tlp x)
         (implies (tlp y)
                  (implies (and (consp x)
                                (equal a (first x)))
                           (in a x))))
```

Example 6

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- **Proof:**

```
Context:
C1: (tlp x)
C2: (tlp y)
C3: (consp x)
C4: a = (first x)

Goal: (in a x)

Proof: ???
```

Example 6

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

- **Proof:**

```
Context:
C1: (tlp x)
C2: (tlp y)
C3: (consp x)
C4: a = (first x)

Goal: (in a x)

Proof:
(in a x)
= { def in }
(consp x) & ( (a = (first x)) | (in a (rest x)) )
= { C3, PL (propositional logic) }
(a = (first x)) | (in a (rest x))
= { C4, PL }
true
```

QED!

Example 7

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

- Claim:

```
(implies (tlp A)
         (implies (tlp B)
                  (implies (and (consp A)
                                (equal z (first A)))
                           (implies (not (in z (aapp A B)))
                                    (not (in z A) )))))
```

Example 7

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

```
Context:
C1: (tlp A)
C2: (tlp B)
C3: (consp A)
C4: z = (first A)
C5: (in z A)

Goal: (in z (aapp A B))

Proof: ???
```

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

Example 7

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

```
Context:
C1: (tlp A)
C2: (tlp B)
C3: (consp A)
C4: z = (first A)
C5: (in z A)
Goal: (in z (aapp A B))
Proof:
(in z (aapp A B))
= { def in }
(consp (aapp A B)) & ( (z = (first (aapp A B)))
                     | (in a (rest (aapp A B))) )
= ...
```

Option 1: expand in

Example 7

```
Context:
C1: (tlp A)
C2: (tlp B)
C3: (consp A)
C4: z = (first A)
C5: (in z A)
Goal: (in z (aapp A B))
Proof:
(in z (aapp A B))
= { def aapp }
(in z (if (endp A) B (cons (car A) (aapp (cdr A) B))))
= { C3, if axioms }
(in z (cons (car A) (aapp (cdr A) B)))
= { def in }
(consp (cons (car A) (aapp (cdr A) B))) &
       ((z = (first (cons (car A) (aapp (cdr A) B)))) |
         (in z (rest (cons (car A) (aapp (cdr A) B)))) ]
= { consp axioms, PL }
       (z = (first (cons (car A) (aapp (cdr A) B)))) |
         (in z (rest (cons (car A) (aapp (cdr A) B)))) ]
= { car-cdr axioms }
       (z = (car A)) |
         (in z (rest (cons (car A) (aapp (cdr A) B)))) ]
= { C4, PL }
true
```

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

Option 2: expand aapp

Did we use all the hypotheses?

Example 7

```
Context:
C1: (tlp A)
C2: (tlp B)
C3: (consp A)
C4: z = (first A)
C5: (in z A)
Goal: (in z (aapp A B))
Proof:
(in z (aapp A B))
= { def aapp }
(in z (if (endp A) B (cons (car A) (aapp (cdr A) B))))
= { C3, if axioms }
(in z (cons (car A) (aapp (cdr A) B)))
= { def in }
(consp (cons (car A) (aapp (cdr A) B))) &
       ((z = (first (cons (car A) (aapp (cdr A) B)))) |
         (in z (rest (cons (car A) (aapp (cdr A) B)))) ]
= { consp axioms, PL }
       (z = (first (cons (car A) (aapp (cdr A) B)))) |
         (in z (rest (cons (car A) (aapp (cdr A) B)))) ]
= { car-cdr axioms }
       (z = (car A)) |
         (in z (rest (cons (car A) (aapp (cdr A) B)))) ]
= { C4, PL }
true
```

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

We didn't have to use C5: what does this imply?

Example 7

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

- We can make our claim stronger:

```
(implies (tlp A)
         (implies (tlp B)
                  (implies (and (consp A)
                                (equal z (first A)))
                           (in z (aapp A B))) ]
```

- Verify this at home!

Example 8

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

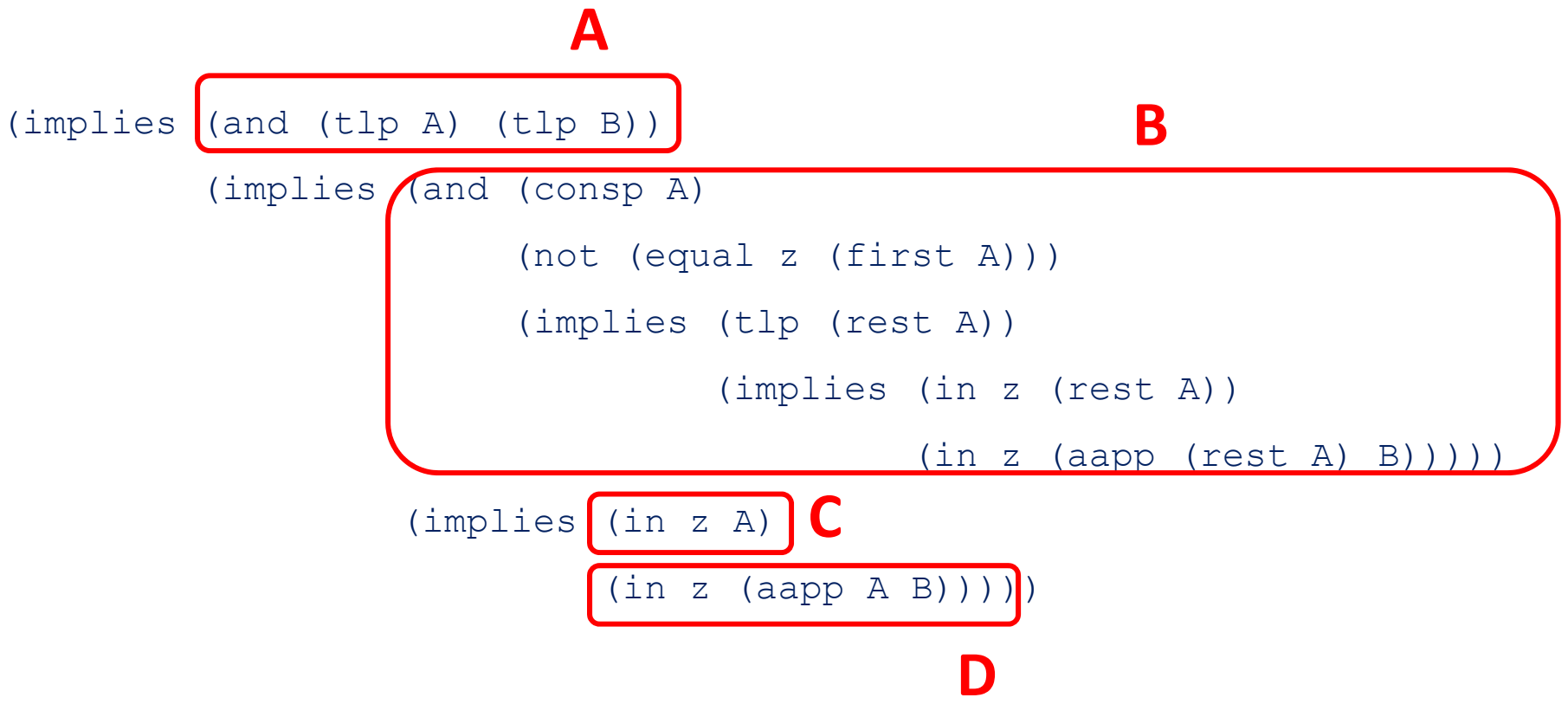
- Claim:

```
(implies (and (tlp A) (tlp B))
         (implies (and (consp A)
                       (not (equal z (first A)))
                       (implies (tlp (rest A))
                                (implies (in z (rest A))
                                         (in z (aapp (rest A) B)))))
                  (implies (in z A)
                           (in z (aapp A B)))))
```

- Wow, this looks complex …
- What do we do?

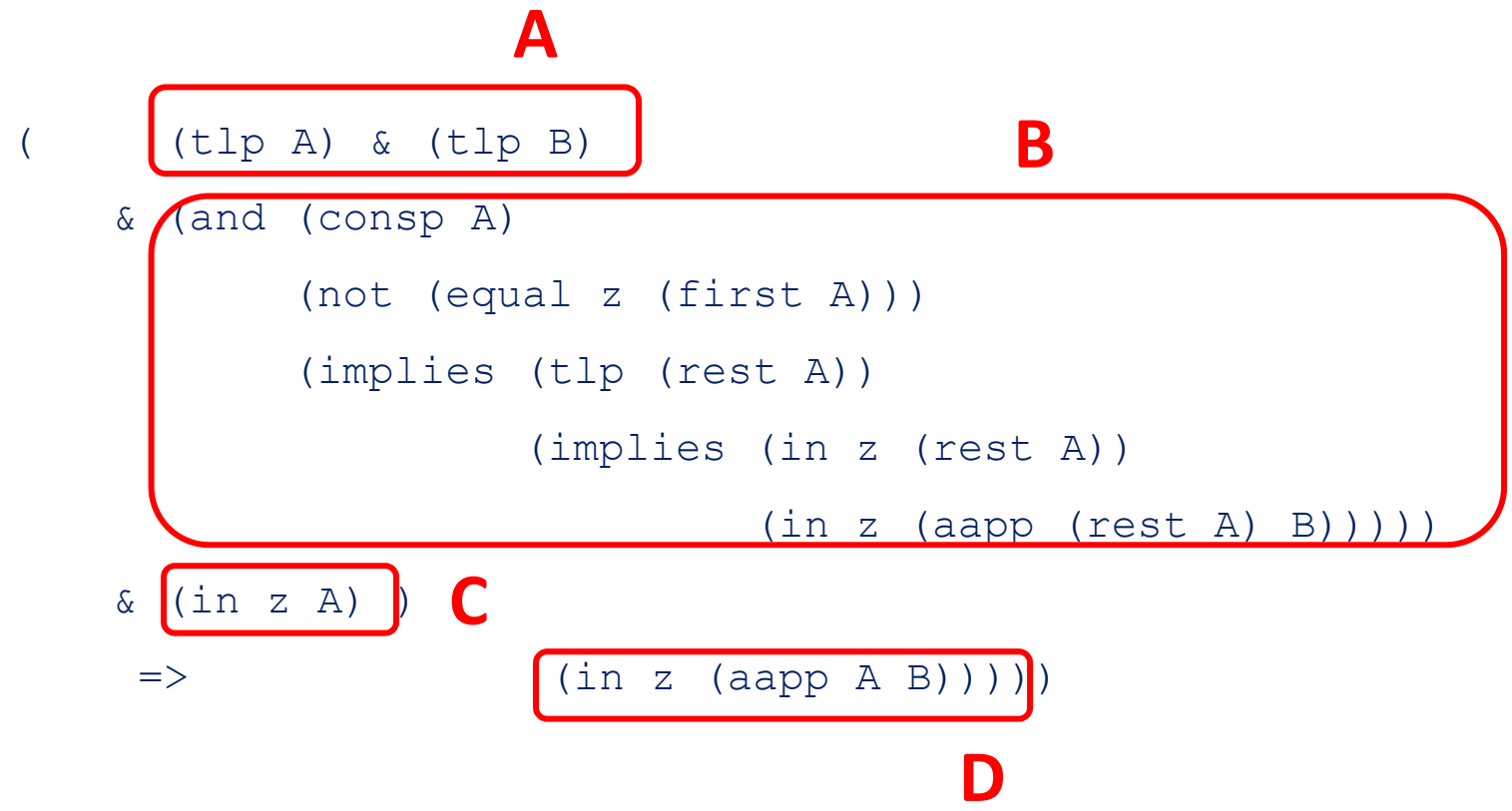
Propositional skeleton to the rescue!

Example 8



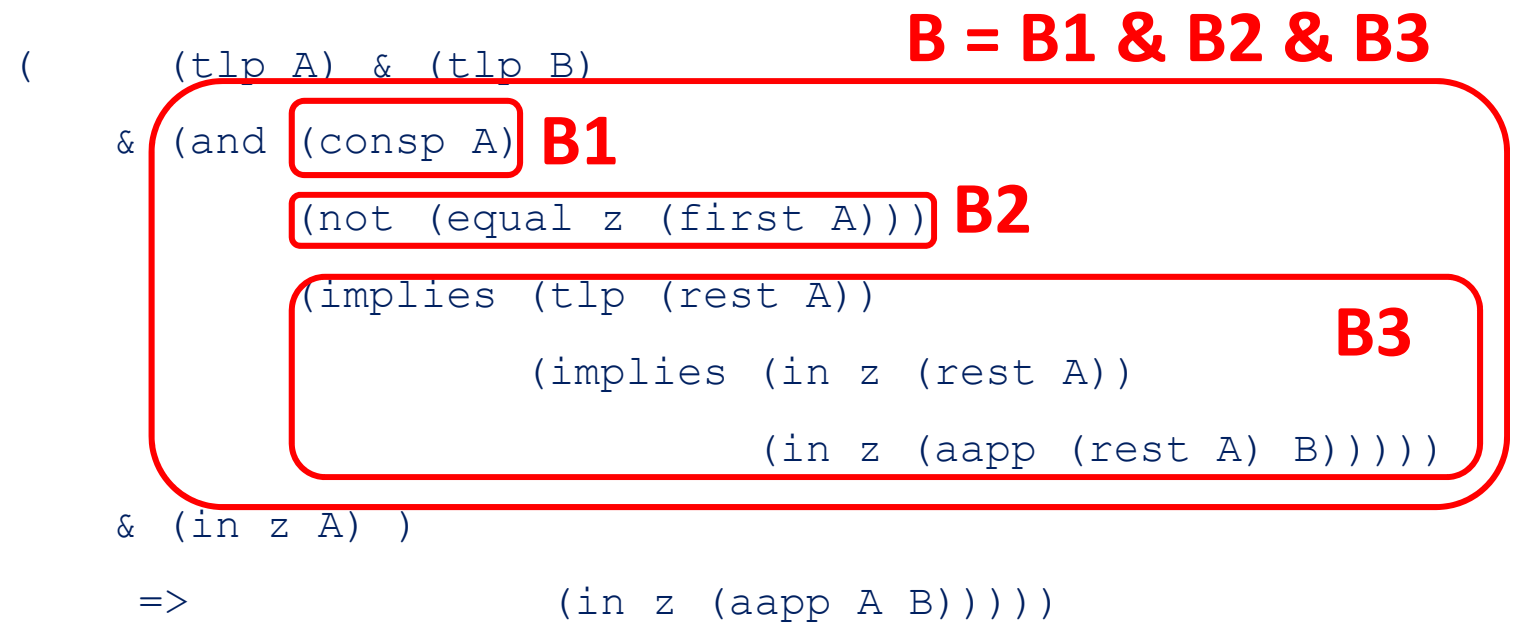
A => (B => (C => D)) = ???

Example 8



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

Example 8


B = B1 & B2 & B3
((tlp A) & (tlp B)
& (and (consp A)
B1
(not (equal z (first A)))
B2
(implies (tlp (rest A))
(implies (in z (rest A))
(in z (aapp (rest A) B)))))
B3
& (in z A))
=> (in z (aapp A B)))))

Example 8

```
(      (tlp A) & (tlp B)
     & (consp A)
     & (not (equal z (first A)))
     & (implies (tlp (rest A))
                (implies (in z (rest A))
                         (in z (aapp (rest A) B)))))
     & (in z A) )
      =>                (in z (aapp A B)))))
```

B1

B2

B3

Example 8

```
(      (tlp A) & (tlp B)
     & (consp A)
     & (not (equal z (first A)))
     & (implies (tlp (rest A))                    B3
                (implies (in z (rest A))
                         (in z (aapp (rest A) B)))))
     & (in z A) )
      =>                (in z (aapp A B)))))
```

Now what?

Example 8

```
(      (tlp A) & (tlp B)
     & (consp A)
     & (not (equal z (first A)))
     & (implies (tlp (rest A))
                (implies (in z (rest A))
                         (in z (aapp (rest A) B))))
     & (in z A) )
      =>                  (in z (aapp A B)))))
```

B3 = X => (Y => Z)

X

Y

Z

B3 = X => (Y => Z) = (X & Y) => Z

Example 8

```
(      (tlp A) & (tlp B)
     & (consp A)
     & (not (equal z (first A)))
     & ( ( (tlp (rest A))
         &(in z (rest A))
          => (in z (aapp (rest A) B)))))
     & (in z A) )
      =>                (in z (aapp A B)))))
```

B3 = (X & Y) => Z)

X

Y

Z

Notice that B3 still contains an implication!

It would be wrong to turn that => into an & !!!

Example 8

```
(     (tlp A) & (tlp B)
    & (consp A)
    & (not (equal z (first A)))
    & ( ( (tlp (rest A))
          &(in z (rest A)) )
           => (in z (aapp (rest A) B)))))
    & (in z A) )
     =>                (in z (aapp A B)))))
```

```
Context:
C1: (tlp A)
C2: (tlp B)
C3: (consp A)
C4: not (z = (first A))
C5: ((tlp (rest A)) & (in z (rest A))) =>
                      (in z (aapp (rest A) B))
C6: (in z A)

Goal: (in z (aapp A B))
```

Example 8

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

```
Context:
C1: (tlp A)
C2: (tlp B)
C3: (consp A)
C4: not (z = (first A))
C5: ((tlp (rest A)) & (in z (rest A))) =>
                        (in z (aapp (rest A) B))
C6: (in z A)

Derived context:
D1: (tlp (rest A)) { C1, C3, def tlp }
D2: (in z (rest A)) { C6, def in, C3, C4, PL }
D3: (in z (aapp (rest A) B)) { C5, D1, D2, MP }

Goal: (in z (aapp A B))

Proof: ???
```

Example 8

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (equal a (first X))
           (in a (rest X)))))
```

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

```
...
C3: (consp A)
C4: not (z = (first A))
C6: (in z A)
...
D3: (in z (aapp (rest A) B)) { C5, D1, D2, MP }

Goal: (in z (aapp A B))

Proof:
(in z (aapp A B))
= { def in, C3, if axioms }
(in z (cons (car A) (aapp (cdr A) B)))
= { def in, consp axioms }
(z = (first A)) | (in z (aapp (cdr A) B))
= { D3, PL }
true
```

QED!

Next time

- Equational reasoning continued