CS313K: Logic, Sets, and Functions

J Strother Moore
Department of Computer Sciences
University of Texas at Austin

Lecture 12 – Chap 4 (4.3, 4.4, 4.5)

Announcements

The mean grade in the midterm was 73. I expected it to be about 83. I conclude that Midterm 1 was too long.

I will curve the grades of Midterm 1 so that the mean is 83.

I will announce the curving mechanism on Thursday. For the moment, just understand that the posted grades are raw scores and they will be adjusted upwards.

```
T1: (equal (rev (app x y))
           (app (rev y) (rev x)))

T2: (iff (true-listp (app x y))
         (true-listp y))

T3: (implies (true-listp x)
             (equal (rev (rev x)) x))
```

T1: (rev (app x y)) = (app (rev y) (rev x))

T2: (true-listp (app x y)) $\leftrightarrow$ (true-listp y)

T3: (true-listp x) $\longrightarrow$ (rev (rev x)) = x

In some of your classes, professors will introduce notation of their own. For example, they might say, "if x and y are sequences then $x \diamond y$ denotes the concatenation of x followed by y, and $\overline{x}$ denotes the reverse of x." They might also say "Let S be the set of all sequences" and assume implicitly that sequences are true-lists. Given such conventions, "$x \in S$" means "x is an element of the set S" or "`(true-listp` x`)`."

T1: $\overline{x \diamond y} = \overline{y} \diamond \overline{x}$.

T2: $(x \diamond y) \in S \leftrightarrow (y \in S)$

T3: $x \in S \rightarrow \overline{\overline{x}} = x$

However it is written, you should understand the logical meaning of these sentences to be:

T1: (rev (app x y)) = (app (rev y) (rev x))

T2: (true-listp (app x y)) $\leftrightarrow$ (true-listp y)

T3: (true-listp x) $\longrightarrow$ (rev (rev x)) = x

Theorem:

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (app (rev b)
                          (rev (rev a)))))
```

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (rev a))))
```

```
(implies (and (true-listp a) (true-listp b))
         (true-listp a))
```

```
(implies (and (true-listp a) (true-listp b))
         t)
```

t

The Rules of Inference are precisely described in Section 4.4.

The reason they're described precisely is so you can learn to do proofs without making mistakes.

I don't care if you learn the "implementation" of the rules. Who cares what π is in your steps? I don't!

But you must learn *how to use* the rules flawlessly and naturally.

Theorem:

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

Transformation 1 (Rewrite: Steps 1 and 2):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

$\uparrow \pi$

Rewrite at π

Transformation 1 (Rewrite: Steps 1 and 2):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

Rewrite at π with

```
T1: (equal (rev (app x y))
           (app (rev y)
                (rev x)))
```

Transformation 1 (Rewrite: Steps 3 and 4):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

Rewrite at π with

```
T1: (implies t                                 ; φ_h
             (equal (rev (app x y))            ; α =
                    (app (rev y)               ; β
                         (rev x))))
```

eqv = equal

σ={x $\leftarrow$ (rev a), y $\leftarrow$ b}

Transformation 1 (Rewrite: Steps 5 and 6):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

Rewrite at π with

```
T1: (implies t                              ; φ_h
             (equal (rev (app x y))         ; α
                    (app (rev y)            ; β
                         (rev x))))
```

eqv = equal

σ={x $\leftarrow$ (rev a), y $\leftarrow$ b}

Prove: ((true-listp a) $\wedge$ (true-listp b)) $\rightarrow$ t

Transformation 1 (Rewrite: Step 7):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (app (rev a) b))))
```

Rewrite at π with

```
T1: (implies t                              ; φ_h
             (equal (rev (app x y))         ; α
                    (app (rev y)            ; β
                         (rev x))))
```

eqv = equal

σ={x $\leftarrow$ (rev a), y $\leftarrow$ b}

β/σ = (app (rev b)
(rev (rev a)))

Transformation 1 (Rewrite: Step 7):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (app (rev b)
                          (rev (rev a)))))
```

Rewrite at π with

```
T1: (implies t                              ; φ_h
             (equal (rev (app x y))         ; α
                    (app (rev y)            ; β
                         (rev x))))
```

eqv = equal

σ={x $\leftarrow$ (rev a), y $\leftarrow$ b}

β/σ = (app (rev b)
                 (rev (rev a)))

Transformation 2:

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (app (rev b)
                          (rev (rev a)))))
```

Transformation 2 (Rewrite: Steps 1,2,3,4,5,6):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (app (rev b)
                          (rev (rev a)))))
```

Rewrite with

```
T2: (iff (true-listp (app x y))      ; α ↔
         (true-listp y))             ; β
```

eqv = iff

σ = {x←(rev b), y←(rev (rev a))}

β/σ = (true-listp (rev (rev a)))

Prove (true-listp a) $\wedge$ (true-listp b) $\longrightarrow$ t

Transformation 2 (Rewrite: Steps 1,2,3,4,5,6):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (rev a))))
```

Rewrite with

```
T2: (iff (true-listp (app x y))    ; α ↔
         (true-listp y))           ; β
```

eqv = iff

σ = $\{$x$\leftarrow$(rev b), y$\leftarrow$(rev (rev a))$\}$

β/σ = (true-listp (rev (rev a)))

Prove (true-listp a) $\wedge$ (true-listp b) $\rightarrow$ t

Transformation 3:

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (rev a))))
```

Transformation 3:

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (rev a))))
```

Transformation 3 (Rewrite: Steps 1,2,3,4,5):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (rev a))))
```

Rewrite with

T3:(implies(true-listp x) ; ϕ_h

(equal (rev (rev x)) x));$\alpha = \beta$

eqv=equal

σ={x $\leftarrow$ a}

β/σ=a

ϕ_h/σ = (true-listp a)

Transformation 3 (Rewrite: Steps 1,2,3,4,5):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp (rev (rev a))))
```

Rewrite with

```
T3:(implies(true-listp x)                    ; φ_h
           (equal (rev (rev x)) x));α = β
```

eqv=equal

σ=$\{$x $\leftarrow$ a$\}$

β/σ=a

ϕ_h/σ = (true-listp a)

Prove (true-listp a) $\wedge$ (true-listp b)

$\longrightarrow$ (true-listp a)

Transformation 3 (Rewrite: Steps 1,2,3,4,5):

```
(implies (and (true-listp a) (true-listp b))
         (true-listp a))
```

Rewrite with

T3:(implies(true-listp x) ; ϕ_h
(equal (rev (rev x)) x));$\alpha = \beta$

eqv=equal

σ={x $\leftarrow$ a}

β/σ=a

ϕ_h/σ = (true-listp a)

Prove (true-listp a) $\wedge$ (true-listp b)
$\rightarrow$ (true-listp a)

Transformation 4:

```
(implies (and (true-listp a) (true-listp b))
         (true-listp a))
```

Transformation 4:

```
(implies (and (true-listp a)   ; α ↔ β (β = t
              (true-listp b))
         (true-listp a))       ; α
```

Use Hyp 1, δ = (true-listp a)

α = (true-listp a), β = t, eqv = iff

Transformation 4:

```
(implies (and (true-listp a)    ; α ↔ β (β = t
              (true-listp b))
         t)                     ; β
```

Use Hyp 1, δ = (true-listp a)

α = (true-listp a), β = t, eqv = iff

Transformation 5:

```
(implies (and (true-listp a)
              (true-listp b))
         t)
```

Taut: `(implies p t)`

σ={p←(and (true-listp a) (true-listp b))}

Transformation 6:

t

```
Thm (implies p t)
Proof:
(implies p t)
=  {rewrite with def implies}
(if p (if t t nil) t)

Case 1: p=nil
(if p (if t t nil) t)
= {by hyp 1}
(if nil (if t t nil) t)
= {by comp}
t
```

```
Case 2: p≠nil  (p ↔ t)
(if p (if t t nil) t)
= {by hyp}
(if t (if t t nil) t)
= {by comp}
t
Q.E.D.
```