CS313K: Logic, Sets, and Functions

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

Lecture 9 – Chap 3 (3.4 – 3.8)

Homework Clarification

On Question 130 you're asked to "list the addresses," where the addresses are things like π_1, π_2, π_3, etc. Just list the *indices* of the addresses, e.g., $1, 2, 3$, etc.

On Implication

```
(defun implies (p q)
  (if p
      (if q t nil)
      t))
```

versus

```
(defun implies (p q)
  (if p
      (if q t nil)
      nil))
```

Let's prove "$x = 3$ implies that $x + 2 = 5$."

First, is it even true? What if x is 7? What if x is -23?

We'll formalize it as:

```
(implies (equal x 3)
         (equal (+ x 2) 5))
```

Is it true for all `x`? Is it true when `x` is `7`? `-23`?

(implies (equal x 3) (equal (+ x 2) 5))

```
(defun implies (p q)
  (if p
      (if q t nil)
      t))
```

versus

```
(defun implies (p q)
  (if p
      (if q t nil)
      nil))
```

Proof

```
(implies (equal x 3)
         (equal (+ x 2) 5))
{hyp}
(implies (equal x 3)
         (equal (+ 3 2) 5))
{computation}
(implies (equal x 3)
         t)
```

Why can we "prove" an implication by assuming the hypothesis is true?

Proof

```
(implies (equal x 3)
         (equal (+ x 2) 5))
{hyp}
(implies (equal x 3)
         (equal (+ 3 2) 5))
{computation}
(implies (equal x 3)
         t)
```

Why can we "prove" an implication by assuming the hypothesis is true? Because the implication is T if the hypothesis is false!

About "Addresses"

In the book I talk about the *address* of an occurrence of one term in another. For example, I might talk about π being the address of the 3rd `A` in

```
(if (zp a) 1 (* a (fact (- a 1))))
```

Some of you want me to tell you what an address *is*. I don't want to, because we don't really care. We just need to have a way to talk about *occurrences*.

```
(if (zp a) 1 (* a (fact (- a 1))))
012345678901234567890123456789012345678
          10        20        30
```

But I could say "an address is the string index of the first character of the occurrence" If I said that, the 3rd a above would have address 27.

But I could, instead, say "an address is $(\alpha\ n)$, where α is a term and n is a number" meaning the n^{th} occurrence of the term α.

There are many other ways I could define *address*.

But how I define address doesn't matter, as long we all understand that they identify an particular occurrence of a term.

About Propositional Occurrences

`(IF x y z)` – `x` is used propositionally but whether `y` and `z` are being used propositionally depends on the context.

In `(IF (IF x y z) 1 2)`, `y` and `z` are being used propositionally.

In `(+ (IF x y z) 0)`, `y` and `z` they are not.

The Propositional Functions

The arguments to `AND`, `OR`, `NOT`, `IMPLIES`, and `IFF` are all used propositionally if the term itself is used propositionally.

So all the variables in

```
(IMPLIES (AND p q) (OR s (NOT r)))
```

are being used propositionally.

I will never use these functions in non-propositional

ways. I.e., I won't write `(+ (AND p q) 3)` even though it is legal.

About Maintaining Equivalence

IFF

```
(implies (and (natp e)

              (mem e x))

         (natp (if (and (f e) (g x))

                   (h (f x))

                   x)))
```

About Maintaining Equivalence

```
IFF       IFF
(implies (and (natp e)

              (mem e x))
         IFF
         (natp (if (and (f e) (g x))

                   (h (f x))

                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF
(implies (and (natp e)
              IFF
              (mem e x))
         IFF
         (natp (if (and (f e) (g x))

                   (h (f x))

                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF   EQ
(implies (and (natp e)
              IFF
              (mem e x))
         IFF
         (natp (if (and (f e) (g x))

                   (h (f x))

                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF
         (natp (if (and (f e) (g x))

                   (h (f x))

                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF   EQ
         (natp (if (and (f e) (g x))

                   (h (f x))

                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF   EQ  IFF
         (natp (if (and (f e) (g x))
                   EQ
                   (h (f x))
                   EQ
                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF   EQ  IFF  IFF   IFF
         (natp (if (and (f e) (g x))
                   EQ
                   (h (f x))
                   EQ
                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF   EQ  IFF  IFFEQ IFFEQ
         (natp (if (and (f e) (g x))
                   EQ EQ EQ
                   (h (f x))
                   EQ
                   x)))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF
         (and (if (and (f e) (g x))

                  (h (f x))

                  x)
              ...))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF  IFF
         (and (if (and (f e) (g x))

                  (h (f x))

                  x)
              ...))
```

About Maintaining Equivalence

```
IFF       IFF  IFF  EQ
(implies (and (natp e)
              IFF EQ EQ
              (mem e x))
         IFF  IFF IFF
         (and (if (and (f e) (g x))
                  IFF
                  (h (f x))
                  IFF
                  x)
              ...))
```