Philosophy 240: Symbolic Logic
Fall 2010
Mondays, Wednesdays, Fridays: 9am - 9:50am

Hamilton College
Russell Marcus
rmarcus1@hamilton.edu

Class 41 - December 8
Functions

I. Characterizing functions

Consider the following argument:

1. No odd numbers are even.
2. One is odd.
3. One is the square of one. / So, not all square numbers are even.

The first two premises are easily regimented into **F**.

1. (x)(Ox ⊃ ~Ex)
2. Oo

We could approach the third premise by using Russell's theory of definite descriptions, using 'Sxo' for 'x is a square of one', and adding a uniqueness clause.

3. (∃x)[Sxo • (y)(Syo ⊃ y=x) • x=o]

We could regiment the conclusion using just monadic predicates:

/ ~(x)[(Sx • Nx) ⊃ Ex]

But, there is a more efficient, and more fecund, option, taking 'the square of x' as a function.
That option will allow us to regiment both the third premise and the conclusion more simply.

We have seen that we can, using the identity predicate, simulate adjectival uses of numbers.
With a small extension of **F**, adding functors like 'f(x)', we can express even more mathematics.
A functor is a symbol used to represent a function, like any of the functions ubiquitous in and essential for mathematics and science.
In mathematics, there are linear function, exponential functions, periodic functions, quadratic functions, and trigonometric functions.
In science, force is a function of mass and acceleration, momentum is a function of mass and velocity, even genetic code is a function.

The utility of functions to mathematics makes them suspect as logic.
Many logic texts, like Hurley, do not include functions since they are considered too mathematical.
But, understanding functions is essential for work in metalogic.
Recall that the semantics for **PL** is presented in terms of truth functions.
All the connectives are truth functions, taking one argument (negation) or two arguments (the rest of the connectives) and yielding a specific truth value.

Consider terms like 'the father of', 'the successor of', 'the sum of', and 'the teacher of'.

Each takes one or more arguments, from their domain, and produces a single output, the range.
One-place functions take one argument, two-place functions take two arguments, n-place functions take n arguments.

Here are some functions, and their logical representations:

the father of:	f(x)
the successor of:	g(x)
the sum of:	f(x,y)
the teacher of:	$g(x_1...x_n)$

The last function can take as arguments, say, all the students in a class.
An essential characteristic of functions is that they yield exactly one value, no matter how many arguments they take.
Thus, the following expressions are not functions:

the parents of a
the classes that a and b share
the square root of x

These expressions are relations; a function is a special type of relation.

By adding functors, we have adopted a new language, which I call **FF**, for Full First-Order Predicate Logic with functors.

II. FF: Syntax and Semantics

Vocabulary of FF

Capital letters A...Z, used as predicates
Lower case letters
- a, b, c, d, e, i, j, k...u are used as constants.
- f, g, and h are used as functors.
- v, w, x, y, z are used as variables.

Five connectives: ~, •, ∨, ⊃ ≡
Quantifier: ∃
Punctuation: (), [], {}

In order to specify the formation rules for **FF**, we have to consider n-tuples of terms.
An **n-tuple of terms** is an ordered series of terms (constants, variables, or functor terms).
'N-tuple' is a general term that covers 'single', 'double', 'triple', 'quadruple', etc.
We use that term since functions can take any number of arguments.
Ordinarily, an n-tuple is represented thus: <a, b, c>
We will represent n-tuples merely by listing the terms separated by commas.

Here are some n-tuples.

a,b
a,a,f(a)
x,y,b,d,f(x),f(a,b,f(x))
a

If α is an n-tuple of terms, then the following are all **functor terms**:

f(α)
g(α)
h(α)

Note that an n-tuple of terms can include functor terms.
'Functor term' is defined recursively, which allows for composition of functions.
For example, one can refer to the grandfather of x, using just the functions for father, e.g. f(x), and mother, e.g. g(x).

f(f(x)) or f(g(x))

The first term represents 'paternal grandfather' and the second represents 'maternal grandfather'.
Similarly, if we take 'h(x)' to represent the square of x, the following represents the eighth power of x, i.e. $((x^2)^2)^2$.

h(h(h(x)))

I have introduced only three functor letters.
As I mentioned regarding variables and constants, there are several different tricks for constructing an indefinite number of terms out of a finite vocabulary, using indexing.
We won't need more than the three letters in this course.
Even with just the three letters, we have an indefinite number of functors, since each of the following is technically a different functor, and can represent a different function.

f(a)
f(a,b)
f(a,b,c)
f(a,b,c,d)
etc.

The scope and binding rules are the same for **FF** as they were for **M** and **F**.
The formation rules only need one small adjustment, at the first line.

Formation rules for wffs of FF.

1. An n-place predicate followed by n terms (constants, variables, **or functor term**s) is a wff.
2. If α is a wff, so are
   $(\exists x)\alpha$, $(\exists y)\alpha$, $(\exists z)\alpha$, $(\exists w)\alpha$, $(\exists v)\alpha$
   $(x)\alpha$, $(y)\alpha$, $(z)\alpha$, $(w)\alpha$, $(v)\alpha$
3. If α is a wff, so is $\sim\alpha$.
4. If α and β are wffs, then so are:
   $(\alpha \cdot \beta)$
   $(\alpha \vee \beta)$
   $(\alpha \supset \beta)$
   $(\alpha \equiv \beta)$
5. These are the only ways to make wffs.

The semantics for **FF** are basically the same as for **F**.
For an interpretation of **FF**, we insert an interpretation of function symbols.

Step 1. Specify a set to serve as a domain of interpretation, or domain of quantification.
Step 2. Assign a member of the domain to each constant.
Step 3. Assign a function with arguments and ranges in the domain to each function symbol.
Step 4. Assign some set of objects in the domain to each one-place predicate; assign sets of ordered n-tuples to each relational predicate
Step 5. Use the customary truth tables for the interpretation of the connectives.

The function assigned in Step 3 will be a function in the meta-language used to interpret the function in the object language.
Iwon't pursue a discussion of meta-linguistic functions, except to say that they work just like ordinary mathematical functions.
Once you have the idea of how functions work in the object language, it will become clear how they work in the meta-language.

III. Translations into FF and simple arithmetic functions

Here are some sentences of **FF**, using the given translation key:

Lxy: x loves y
f(x): the father of x
g(x): the mother of x
o: Olaf

Olaf loves his mother:	Log(o)
Olaf loves his grandmothers:	Log(g(o)) • Log(f(o))
Noone is his/her own mother:	$(x)\sim x=g(x)$

Many simple concepts in arithmetic are functions: addition, multiplication, least common multiple.
The most fundamental function in mathematics is the successor function.
All other mathematical functions can be defined in terms of successor and other basic concepts.

In fact, all of arithmetic can be developed from five basic axioms, called the Peano axioms.
They are named for Giuseppe Peano, who published in 1889 a precise version of the axioms that Richard Dedekind had published a year earlier.
(Peano had credited Dedekind, and sometimes these axioms are called the Dedekind-Peano, or even the Dedekind, axioms.)

Peano's Axioms for Arithmetic (following Mendelson, with adjustments)

The simplified regimentations of the axioms on the right, use:

a: zero; Nx: x is a number; f(x): the successor of x

P1: 0 is a number	P1. Na
P2: The successor (x') of every number (x) is a number	P2. $(x)(Nx \supset Nf(x))$
P3: 0 is not the successor of any number	P3. $\sim(\exists x)(Nx \bullet f(x)=a)$
P4: If x'=y' then x=y	P4. $(x)(y)[(Nx \bullet Ny) \supset (f(x)=f(y) \supset x=y)]$
P5: If P is a property that may (or may not) hold for any number, and if i. 0 has P; and ii. for any x, if x has P then x' has P; then all numbers have P.	P5. $\{Pa \bullet (x)[(Nx \bullet Px) \supset Pf(x)]\} \supset (x)(Nx \supset Px)$

P5 is also called the induction schema, and is actually a schema of an infinite number of axioms.
Mathematical induction is essential in advanced logic, as well as in linear algebra and number theory.

Here are a few translations of arithmetic sentences using functions.
Note: in the following sentences, take 'number' to mean 'natural number'.

o: one	Ex: x is even
f(x): the successor of x	Ox: x is odd
f(x, y): the product of x and y	Px: x is prime

1. One is not the successor of any number.
   $(x)(Nx \supset \sim f(x)=o)$

2. If the product of a pair of numbers is odd, then the product of the successors of those numbers is even.
   $(x)(y)\{(Nx \bullet Ny) \supset [Of(x, y) \supset Ef(f(x), f(y))]\}$

3. There are no prime numbers such that their product is prime.
   $\sim(\exists x)(\exists y)[Nx \bullet Px \bullet Ny \bullet Py \bullet Pf(x, y)]$

IV. Exercises A. Use the given key to translate the following sentences into **FF**.

o: one
t: two
f(x): the successor of x
g(x): the square of x
f(x, y): the product of x and y
g(x, y): the sum of x and y
Ex: x is even
Nx: x is a natural number (i.e. 1, 2, 3...)
Ox: x is odd
Px: x is prime

Note: in the following sentences, take 'number' to mean 'natural number'.

1. One is not prime, but its successor is.
2. Every number has a successor.
3. The successor of an odd number is even.
4. If the sum of a pair of numbers is even then either both members of the pair are even or both members are odd.
5. If the sum of a pair of numbers is odd, then one member of the pair is odd and the other member is even.
6. The product of a pair of prime numbers is not prime.
7. The square of an even number is even and the square of an odd number is odd.
8. The successor of the square of an even number is odd.
9. The sum of two and a prime number other than two is odd.
10. There is a pair of distinct prime numbers such that their product is the successor of their sum.

V. Derivations using functions

There are no new rules covering functions, which act like simple terms.
Since a function always produces a single element from the domain, no matter how many arguments it takes, we can consider a functor as if it were either a constant or a variable.
Whether we should treat a function as a constant or a variable, for the purposes of UG and EI depends on what the arguments of the function are.
We can, for example, UI to a variable, or a function of a variable, or any complex function all of whose arguments are variables.
For UG, if the arguments of a function are all variables, then we are free to use UG over the variables in that function; If the arguments contain any constants, then we can not use UG.
For EI, we must continue always to instantiate to a new term.
A functor is not a new term if any of its arguments, or any of the arguments of any of its sub-functors, have already appeared in the derivation.

Let's return to the argument in §I:

1. No odd numbers are even.
2. One is odd.
3. One is the square of one. / So, not all square numbers are even.

We can use a function (f(x): the square of x) to regiment the third premise and conclusion more simply.

1. (x)(Ox ⊃ ~Ex)
2. Oo
3. o=f(o) / ~(x)Ef(x)

And now we can derive the conclusion:

4. Of(o) 2, 3, ID
5. Of(o) ⊃ ~Ef(o) 1, UI
6. ~Ef(o) 5, 4, MP
7. (∃x)~Ef(x) 6, EG
8. ~(x)Ef(x) 7, CQ

QED

Here is a derivation which uses some composition of functions.
Note that B is a two-place predicate, taking as arguments a variable and a functor term with a variable argument in the first premise, and taking as arguments two functor terms, each with variable arguments, in the conclusion.

1. (x)[Ax ⊃ Bxf(x)]
2. (∃x)Af(x) / (∃x)Bf(x)f(f(x))
3. Af(a) 2, EI (to ‘a’)
4. Af(a) ⊃ Bf(a)f(f(a)) 1, UI (to ‘f(a)’)
5. Bf(a)f(f(a)) 4, 3, MP
6. (∃x)Bf(x)f(f(x)) 5, EG

QED

In the following short derivation, we instantiate to a two-place function, f(x, g(x)), one of whose arguments is itself a function.

1. ~(∃x)Cx / (x)~Cf(x, g(x))
2. (x)~Cx 1, CQ
3. ~Cf(x, g(x)) 2, UI
4. (x)~Cf(x, g(x)) 3, UG

QED

Note that since none of the arguments of any of the functions above are constants, UG is permissible.

Here is a slightly longer argument:

1. (x){(Nx • Gxt) ⊃ (∃y)(∃z)[Py • Pz • x=f(y, z)]}		
2. Nb • Gbt	/ (∃x)(∃y)(∃z)[Nx • Py • Pz • x=f(y, z)]	
3. (Nb • Gbt) ⊃ (∃y)(∃z)[Py • Pz • b=f(y, z)]		1, UI
4. (∃y)(∃z)[Py • Pz • b=f(y, z)]		3, 2, MP
5. (∃z)[Pa • Pz • b=f(a, z)]		4, EI
6. Pa • Pc • b=f(a, c)		5, EI
7. Nb		2, Simp
8. Nb • Pa • Pc • b=f(a, c)		7, 6, Conj
9. (∃z)[Nb • Pa • Pz • b=f(a, z)]		8, EG
10. (∃y)(∃z)[Nb • Py • Pz • b=f(y, z)]		9, EG
11. (∃x)(∃y)(∃z)[Nx • Py • Pz • x=f(y, z)]		10, EG

QED

Note that this last argument is given a natural interpretation as follows:

> All natural numbers greater than two are equal to the sum of two primes.
> A billion is a natural number greater than two.
> Therefore, some natural number is equal to the sum of two primes.

The first premise in this argument is a famously unproven statement called Goldbach's conjecture.

VI. Exercises B. Derive the conclusions of each of the following arguments.

1. 1. (x)[Ax ∨ Bf(x)] / (x)[Af(x) ∨ Bf(f(x))]

2. 1. (x)(y)[f(x)=y ⊃ Cyxc]
   2. ~Cf(a)ac / f(a)≠b

3. 1. (x){Dx ⊃ (y)[~Exy ≡ Gf(f(y))]}
   2. (x)(Dx • ~Gx) / (x)Ef(x)f(x)

Solutions may vary.

VII. Solutions

Answers to Exercises A:

1. ~Po • Pf(o)
2. (x)[Nx ⊃ (∃y)y=f(x)]
3. (x)(y)[(Ny • Oy • x=f(y)) ⊃ Ex]
4. (x)(y){(Nx • Ny) ⊃ {Eg(x, y) ⊃ [(Ex • Ey) ∨ (Ox • Oy)]}}
5. (x)(y){(Nx • Ny) ⊃ {Og(x, y) ⊃ [(Ex • Oy) ∨ (Ox • Ey)]}}
6. (x)(y)[(Nx • Px • Ny • Py) ⊃ ~Pf(x, y)]
7. (x)[(Nx • Ex) ⊃ Eg(x)] • (x)[(Nx • Ox) ⊃ Og(x)]
8. (x)[(Nx • Ex) ⊃ Of(g(x))]
9. (x)[(Nx • Px • ~x=t) ⊃ Og(t, x)]
10. (∃x)(∃y)[Nx • Px • Ny • Py • ~x=y • f(x, y)=f(g(x, y))]

Sample answers to Exercises B:

1.
1.	(x)[Ax ∨ Bf(x)]	/ (x)[Af(x) ∨ Bf(f(x))]
2.	Af(x) ∨ Bf(f(x))	1, UI
3.	(x)[Af(x) ∨ Bf(f(x))]	2, UG

QED

2.
1.	(x)(y)[f(x)=y ⊃ Cyxc]	
2.	~Cf(a)ac	/ f(a)≠b
| 3.	f(a)=b	AIP
| 4.	(y)[f(a)=y ⊃ Cyac]	UI, 1
| 5.	f(a)=b ⊃ Cbac	UI, 4
| 6.	Cbac	5, 3, MP
| 7.	Cf(a)ac	6, 3, ID
| 8.	Cf(a)ac • ~Cf(a)ac	7, 2, Conj
9.	f(a)≠b	

QED

3.
1.	(x){Dx ⊃ (y)[~Exy ≡ Gf(f(y))]}		
2.	(x)(Dx • ~Gx)	/ (x)Ef(x)f(x)	
3.	Df(x) • ~Gf(x)	2, UI	
4.	Df(x) ⊃ (y)[~Ef(x)y ≡ Gf(f(y))]	1, UI	
5.	Df(x)	3, Simp	
6.	(y)[~Ef(x)y ≡ Gf(f(y))]	4, 5, MP	
7.	Df(f(f(x))) • ~Gf(f(f(x)))	2, UI	
8.	~Ef(x)f(x) ≡ Gf(f(f(x)))	6, UI	
9.	[~Ef(x)f(x) ⊃ Gf(f(f(x)))] • [Gf(f(f(x))) ⊃ ~Ef(x)f(x)]	8, Equiv	
10.	~Ef(x)f(x) ⊃ Gf(f(f(x)))	9, Simp	
11.	~Gf(f(f(x)))	7, Com, Simp	
12.	Ef(x)f(x)	10, 11, MT, DN	
13.	(x)Ef(x)f(x)	12, UG	

QED