Presheaves-Calc

June 4, 2020

[1]:
```
from itertools import product


# Source and target are part of the data of a morphism
# Different morphisms should have different names, except 'identity' morphisms
class Morphism:
    def __init__(self, name, source, target):
        self.name = name          # name: each morphism should have
                                  # a different name (a string)
                               # except identity morphisms that are all called '1'
        self.source = source      # string representing source object
        self.target = target      # string representing target object

# Identity morphisms are all called '1'
def not_identity(morphism):
    return morphism.name != '1'


class Category:
    def __init__(self, objects, morphisms,composition):
        self.objects = objects        # list of strings giving the objects
        self.morphisms = morphisms    # list of the non-identity Morphisms's
        for C in objects:
            identity = Morphism('1',C,C)
            self.morphisms.append(identity) # Add identity Morphism's
                                            # for each object.

        self.composition = composition      # Composition is a dictionary:
                                            # key = pair of Morphism names,
                                            # value = name of composition.

       for rho in morphisms:             # Add composition for identity morphisms
            r = rho.name
            self.composition[(r,'1')] = r
            self.composition[('1',r)] = r
```

```python
# Mathematical functions will be represented by python dictionaries.
# If these functions appear as sections of a presheaf, then they need a name.
# So we will define a class "Named Dictionary".
class Named_Dictionary:
    def __init__(self,dictionary):
        self.dictionary = dictionary
        self.name = str(dictionary)
   def __hash__(self):   # to use a mathematical function as key for a dictionary
        return hash(self.name)
    def __getitem__(self,item):     # function application
        return self.dictionary[item]

# for lists X and Y, hom(X,Y) returns a list of functions from X to Y
# as dictionaries
def hom(X,Y):
    return [dict(zip(X,y)) for y in product(Y,repeat=len(X))]

# F,G are presheaves on a Category cat, each represented by a dictionary, with
# key = name of the object, value = list of sections
# (e.g. strings or Named_Dictionaries)
# OR key = name of morphism in cat,
# value = dictionary representing the restriction map.
# f is a function from F to G, represented by a dictionary, with
# key = name of the object (say 'C'), value = function from F['C'] to G['C'].
def is_morphism(f,F,G,cat):
    cat_nontriv_morph = [f for f in cat.morphisms if not_identity(f)]
    for rho in cat_nontriv_morph:
        D = rho.source
        C = rho.target
        r = rho.name
        f_C = f[C]
        f_D = f[D]
        for s in F[C]:
            if f_D[F[r][s]] != G[r][f_C[s]]:
                return False
    return True

# F,G are presheaves on Category cat.
# Hom_PSh(F,G,cat) gives a list of morphisms from F to G.
# Each morphism is represented as a dictionary, with
# key = name of the object (say 'C'), value = function from F['C'] to G['C'].
def Hom_PSh(F,G,cat):
    HOM = {}
    for C in cat.objects:
        HOM[C] = hom(F[C],G[C])
    keys, values = zip(*HOM.items())
    maps = [dict(zip(keys, v)) for v in product(*values)]
```

```
    psh_morphisms = [f for f in maps if is_morphism(f,F,G,cat)]
    return psh_morphisms

# F,G are presheaves on Category cat.
# Product_PSh(F,G,cat) gives the product presheaf of F and G.
# Sections of the product presheaf are named (a,b),
# with a and b the name of a section in F resp. G.
def Product_PSh(F,G,cat):
    FxG = {}
    for C in cat.objects:
        FxG[C] = list(product(F[C],G[C]))
    for rho in filter(not_identity, cat.morphisms):
        r = rho.name
        C = rho.target
        FxG[r] = {}
        for a,b in product(F[C],G[C]):
            FxG[r][(a,b)] = (F[r][a],G[r][b])
    return FxG

# Sum_PSh(F,G,cat) gives the coproduct presheaf of F and G
# (both presheaves on Category cat).
# The name of the section is the same as the name
# of the corresponding section in F or G.
def Sum_PSh(F,G,cat):
    sum_psh = {}
    for C in cat.objects:
        sum_psh[C] = F[C]+G[C]
    for rho in filter(not_identity, cat.morphisms):
        r = rho.name
        sum_psh[r] = {}
        for s in F[C]:
            sum_psh[r][s] = F[r][s]
        for s in G[C]:
            sum_psh[r][s] = G[r][s]
    return sum_psh

# Terminal(cat) returns the terminal presheaf on the Category cat.
def Terminal(cat):
    terminal = {}
    for C in cat.objects:
        terminal[C] = ['*']
    for rho in filter(not_identity, cat.morphisms):
        r = rho.name
        terminal[r] = {}
        terminal[r]['*'] = '*'
    return terminal
```

```python
# Yoneda(C,cat) returns the representable presheaf represented
# by the object C of the Category cat
def Yoneda(C,cat):
    yC = {}
    for D in cat.objects:
       yC[D] = [f.name for f in cat.morphisms if f.source == D and f.target == C]
    for rho in filter(not_identity,cat.morphisms):
        r = rho.name
        D = rho.target
        comp = cat.composition
        yC[r] = {}
        for f in yC[D]:
            yC[r][f] = comp[(f,r)]
    return yC

# Internal_Hom(F,G,cat) returns a the internal Hom from F to G
# in the topos of presheaves on Category cat.
# Sections are represented by Named_Dictionary's.
def Internal_Hom(F,G,cat):
    H = {}
    for C in cat.objects:
        yC = Yoneda(C,cat)
        yCxF = Product_PSh(yC,F,cat)
        H[C] = [Named_Dictionary(f) for f in Hom_PSh(yCxF,G,cat)]
    for rho in filter(not_identity,cat.morphisms):
        r = rho.name
        C = rho.target
        D = rho.source
        yC = Yoneda(C,cat)
        yD = Yoneda(D,cat)
        yCxF = Product_PSh(yC,F,cat)
        yDxF = Product_PSh(yD,F,cat)
        H[r] = {}
        # s is a Named_Dictionary.
        # s.dictionary has as key an object E of cat, and
        # as value a dictionary going from elements
        # of yCxF[E] to elements of G[E]
        for s in H[C]:
            restr_s = {}
            for E in s.dictionary.keys():
                restr_s[E] = {}
                for a,x in yDxF[E]:
                    b = cat.composition[(r,a)]
                    restr_s[E][(a,x)] = s.dictionary[E][(b,x)]
            H[r][s]=Named_Dictionary(restr_s)
    return H
```

```python
# inverse_image(F,phi,Ccal,Dcal) returns the inverse image of F along f,
# where f is the geometric morphism f : PSh(Ccal) --> PSh(Dcal)
# induced by a functor phi : Ccal --> Dcal.
# The functor phi is given by a dictionary, with
# key = object/morphism of Ccal, value = corresponding object/morphism of Dcal
def inverse_image(F,phi,Ccal,Dcal):
    phi_star_F = {}   # F is sheaf on Dcal, phi_star_F is sheaf on Ccal
    for C in Ccal.objects:
        phi_star_F[C] = F[phi[C]]
    for rho in filter(not_identity,Ccal.morphisms):
        r = rho.name
        phi_star_F[r] = F[phi[r]]
    return phi_star_F
```

[2]:
```python
# Example 1: directed graphs as presheaves.

V = 'V'
E = 'E'

source = Morphism('source',V,E)
target = Morphism('target',V,E)

Ccal = Category([V,E],[source,target],{})

F = {}
F[V] = ['a','b']
F[E] = ['alpha']
F[source.name] = { 'alpha' : 'a' }
F[target.name] = { 'alpha' : 'a' }

G = {}
G[V] = ['x','y']
G[E] = ['beta']
G[source.name] = { 'beta' : 'x' }
G[target.name] = { 'beta' : 'y'}
```

[3]:
```python
Hom_PSh(G,F,Ccal)
```

[3]:
```
[{'V': {'x': 'a', 'y': 'a'}, 'E': {'beta': 'alpha'}}]
```

[4]:
```python
FxG = Product_PSh(F,G,Ccal)
FxG
```

[4]:
```
{'V': [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b', 'y')],
 'E': [('alpha', 'beta')],
 'source': {('alpha', 'beta'): ('a', 'x')},
 'target': {('alpha', 'beta'): ('a', 'y')}}
```

[5]:
```python
len(Hom_PSh(FxG,G,Ccal))
```

[5]: 4

[6]:
```python
yV = Yoneda('V',Ccal)
print(yV)
yE = Yoneda('E',Ccal)
print(yE)
terminal = Terminal(Ccal)
print(terminal)
```

```
{'V': ['1'], 'E': [], 'source': {}, 'target': {}}
{'V': ['source', 'target'], 'E': ['1'], 'source': {'1': 'source'}, 'target':
{'1': 'target'}}
{'V': ['*'], 'E': ['*'], 'source': {'*': '*'}, 'target': {'*': '*'}}
```

[7]:
```python
H = Internal_Hom(F,G,Ccal)
for C in Ccal.objects:
    print(C, ":",len(H[C]))
```

```
V : 4
E : 4
```

[8]:
```python
# Example 2 : reflexive directed graphs as presheaves

M = 'M'

source = Morphism('source',M,M)
target = Morphism('target',M,M)

composition_in_M = {}
composition_in_M['source','source'] = 'source'
composition_in_M['source','target'] = 'source'
composition_in_M['target','source'] = 'target'
composition_in_M['target','target'] = 'target'

Dcal = Category([M],[source,target],composition_in_M)


# define functor phi : Ccal to Dcal
phi = {}
phi[V] = M
phi[E] = M
phi['source'] = 'source'
phi['target'] = 'target'
```

[9]:
```
yM = Yoneda(M,Dcal)
yM
```

[9]:
```
{'M': ['source', 'target', '1'],
 'source': {'source': 'source', 'target': 'target', '1': 'source'},
 'target': {'source': 'source', 'target': 'target', '1': 'target'}}
```

[10]:
```
inverse_image(yM,phi,Ccal,Dcal)
```

[10]:
```
{'V': ['source', 'target', '1'],
 'E': ['source', 'target', '1'],
 'source': {'source': 'source', 'target': 'target', '1': 'source'},
 'target': {'source': 'source', 'target': 'target', '1': 'target'}}
```