A Rudimentary Proof Assistant for Categorical Logic in Python

Clarence Protin

December 29, 2017

1 Introduction

It is interesting to view categories from the perspective of the internal category concept. This suggests a programming interpretation which is also constructivist: arrows and objects can be seen as instances of their corresponding classes. We consider the formation of products as another kind of class which includes ordinary objects as a field and whose constructor initializes object and arrow instances. We also could consider as part of the constructor the updating of a global equation list which specifies the equations which hold for the created objects and arrows of the current environment, referenced by corresponding name tags (see our code example in the link below for more details). We have thus also a rudimentary form of proof assistant in which the user must create instances of the object class and form products and compositions from earlier data, in particular data automatically created in the creation of products. The universal properties are captured by using the functional *hom*-set interpretation and by processing equations. Here

`https://drive.google.com/file/d/1H_iizyBAHBJbvhjatL5PT4fuIBwZNk_a/view?usp=sharing`

is a simple program illustrating some of the above ideas for the *positive intuitionistic propositional calculus* of Lambek and Scott (i.e. we have only $\wedge$ and $\rightarrow$) which strives for the middle ground between logic and category theory. Besides a direct extension to the full intuitionistic propositional calculus, simply typed lambda calculus and cartesian closed categories could be obtained by extending the equational aspect of this program. Here is an example of the derivation of rules R'4b and R'4c of Lambek and Scott p.49.

```
$ python -i deduction.py

Welcome to CatLog v0.1

type help() for command list

added: T
added: id_T : T -> T
>>> help()
Obj(<name>) - creates an object
Comp(<name1>,<name2>) - composes two arrows
Conj(<name1>,<name2>) - creates conjunction object with associated morphisms
Join(<name1>,<name2>) - <f,g> - see Lambek and Scott p.48
Imp(<name1>,<name2>) -  internal implication <name1> <= <name2>
Trans(<name>) - transpose f* - see Lambek and Scott p.49
Hyp(<arrowname>,<source>,<target>) - introduces a hypothesis (creates an arrow)
disp() - displays current objects, arrows and history
equations() - displays category equations (Work in Progress)
Sym(n) - adds B = A if A = B is equation n
Tr(n,m) - adds A = C if equation n is A = B and equation m is B = C
Sub(n,m,t) - (Work in Progress)
clear() - clear current environment
>>> Obj("C")
added: C
added: id_C : C -> C
added: O_C : C -> T
>>> Obj("B")
added: B
added: id_B : B -> B
added: O_B : B -> T
>>> Conj("C","B")
```

```
Conjunction of C and B
added: (C /\ B)
added: id_(C /\ B) : (C /\ B) -> (C /\ B)
added: O_(C /\ B) : (C /\ B) -> T
added: p1_(C /\ B) : (C /\ B) -> C
added: p2_(C /\ B) : (C /\ B) -> B
>>> Trans("id_(C /\ B)")
Implication of (C /\ B) and B
added: ((C /\ B) <= B)
added: id_((C /\ B) <= B) : ((C /\ B) <= B) -> ((C /\ B) <= B)
added: O_((C /\ B) <= B) : ((C /\ B) <= B) -> T
added: (id_(C /\ B))* : C -> ((C /\ B) <= B)
>>> disp()
Objects:

T
C
B
(C /\ B)
((C /\ B) <= B)

Arrows:

id_T : T -> T
id_C : C -> C
O_C : C -> T
id_B : B -> B
O_B : B -> T
id_(C /\ B) : (C /\ B) -> (C /\ B)
O_(C /\ B) : (C /\ B) -> T
p1_(C /\ B) : (C /\ B) -> C
p2_(C /\ B) : (C /\ B) -> B
id_((C /\ B) <= B) : ((C /\ B) <= B) -> ((C /\ B) <= B)
O_((C /\ B) <= B) : ((C /\ B) <= B) -> T
(id_(C /\ B))* : C -> ((C /\ B) <= B)

Rules:

Created T
Created C
Created B
Conjunction of C and B
Created (C /\ B)
Implication of (C /\ B) and B
Created ((C /\ B) <= B)
Transposition of id_(C /\ B) : (C /\ B) -> (C /\ B)
>>> clear()
added: T
added: id_T : T -> T
>>> Obj("D")
added: D
added: id_D : D -> D
added: O_D : D -> T
>>> Obj("A")
added: A
added: id_A : A -> A
added: O_A : A -> T
>>> Hyp("g","D","A")
Hypothesis added: g : D -> A
>>> Obj("B")
added: B
```

```
added: id_B : B -> B
added: O_B : B -> T
>>> Imp("D","B")
Implication of D and B
added: (D <= B)
added: id_(D <= B) : (D <= B) -> (D <= B)
added: O_(D <= B) : (D <= B) -> T
>>> Conj("(D <= B)", "B")
Conjunction of (D <= B) and B
added: ((D <= B) /\ B)
added: id_((D <= B) /\ B) : ((D <= B) /\ B) -> ((D <= B) /\ B)
added: O_((D <= B) /\ B) : ((D <= B) /\ B) -> T
added: p1_((D <= B) /\ B) : ((D <= B) /\ B) -> (D <= B)
added: p2_((D <= B) /\ B) : ((D <= B) /\ B) -> B
added: e_D,B : ((D <= B) /\ B) -> D
>>> Comp("e_D,B","g")
added: e_D,B o g : ((D <= B) /\ B) -> A
>>> Trans("e_D,B o g")
Implication of A and B
added: (A <= B)
added: id_(A <= B) : (A <= B) -> (A <= B)
added: O_(A <= B) : (A <= B) -> T
added: (e_D,B o g)* : (D <= B) -> (A <= B)
>>> disp()
Objects:

T
D
A
B
(D <= B)
((D <= B) /\ B)
(A <= B)

Arrows:

id_T : T -> T
id_D : D -> D
O_D : D -> T
id_A : A -> A
O_A : A -> T
g : D -> A
id_B : B -> B
O_B : B -> T
id_(D <= B) : (D <= B) -> (D <= B)
O_(D <= B) : (D <= B) -> T
id_((D <= B) /\ B) : ((D <= B) /\ B) -> ((D <= B) /\ B)
O_((D <= B) /\ B) : ((D <= B) /\ B) -> T
p1_((D <= B) /\ B) : ((D <= B) /\ B) -> (D <= B)
p2_((D <= B) /\ B) : ((D <= B) /\ B) -> B
e_D,B : ((D <= B) /\ B) -> D
e_D,B o g : ((D <= B) /\ B) -> A
id_(A <= B) : (A <= B) -> (A <= B)
O_(A <= B) : (A <= B) -> T
(e_D,B o g)* : (D <= B) -> (A <= B)

Rules:

Created T
Created D
Created A
```

```
Created B
Implication of D and B
Created (D <= B)
Conjunction of (D <= B) and B
Created ((D <= B) /\ B)
Composed e_D,B and g
Implication of A and B
Created (A <= B)
Transposition of e_D,B o g : ((D <= B) /\ B) -> A
```

The Arrows list can be seen as a proof of its last element applying the rules. The creation of arrows corresponds to introducing hypotheses. We leave it as an exercise to illustrate the deduction theorem for our program.