#4
AUTOLOG CATEGORY METALOGIC
8/16/2019

§1 Autolog metalogic for Category Algebraic Types -- CATs

There would be many ways to express category theory(s) in autolog. The style chosen here emphasizes an interesting mixture of category notions and type notations. As with the previous lectures, the type notations have the main purpose of establishing an indexicality regimen for the inference rules and equality modulators.

Here are some basic inference forms (rules and modulators) for CAT:

```
«
     true => cat:type.          //                        1
     C:cat => obj(C):type.      // objects                2
     X:obj(C), Y:obj(C) =>      // maps                   3
            X→Y:type.
     X:obj(C), Y:obj(C), Z:obj(C), // composition         4
          F:X→Y, G:Y→Z => F◦G:X→Z.
     (F◦G)◦H = F◦(G◦H).              // associativity      5
     %% identity maps
     X:obj(C) => id(X):X→X.                    //          6
     X:obj(C), Y:obj(C), F:X→Y => id(X)◦F=F. //           7
     X:obj(C), Y:obj(C), F:X→Y => F◦id(Y)=F. //           8
»
```

Rule 2 makes obj(C) a dependent type. In rule 3 this makes X→Y also a dependent type implicitly. This would be an example of the indexicality of the consequent

literal: X and Y are indexed in the antecedent, and that indexing forces the dependence on C. One could force the implicit typing to look explicit like this:

«

```
X:obj(C), Y:obj(C) =>     // dependence on C     3'
       X->Y(C):type.
hom(X,Y,C)=X→Y(C).    // 2 notations, same type
```

»

See the previous lectures for the definition (and examples) of autolog indexicality and various typing notational conventions.

Notice that the modulator for associativity of ◦ (in 5) is also implicitly typed, assuming that ◦ is only used in the theory for composition of morphisms.

{ The « » notation encloses autolog source code and forces all outside text to be our commentary. Compiling the text lecture from the autolog editor will compile only the « » included sections. }

§2 Morphism definional types

A "monomorphism" is a morphism which is left cancellable for composition. The autolog definition amounts to two rules, one unfolding the meaning of "mono" and one supplying a condition for concluding that a morphism is mono. For example,

«

```
F:X→Y, F:mono, G:Y→Z, H:Y→Z, F◦G=F◦H => G=H.
F:X→Y,  ∀((G:Y→Z) ∧ (H:Y→Z), (F◦G=F◦H)→(G=H))
              => F:mono.
```

»

As in Lecture #2 and #3, the ∀-form serves as a "lemma":

Prove the lemma in order to conclude that F is mono. Establishing such a ∀-lemma is illustrated in previous lectures.

EXERCISE A. Give a similar definition for "epimorphism" (right cancellable morphism).

Then the bimorphism type could be defined as

«

```
bi = (mono' x 'epi).
```

»

Recall from Lecture #3 that x (product) types would satify the type modulator

«

```
A:T' x 'S = (A:T)∧(A:S).
```

»

An "isomorphism" has an existential definition

«

```
F:X→Y, F:iso => G:Y→X, F∘G=id(X), G∘F=id(Y).
F:X→Y, G:Y→X, F∘G=id(X), G∘F=id(Y) => F:iso.
```

»

which is conveniently defined usuing a coherent logic form without needing to introduce an ∃-lemma.

EXERCISE B. Using similar autolog expressions, define "endomorphism" type using rules and "automorphism" type in terms of endomorphism and isomorphism using a type modulator.

§3 Autolog Functor metalogic

A "functor" type depends on (is indexed via) two categorie a source C and a target D.

«

```
     C:cat, D:cat => fnctr(C,D):type.
     F:fnctr(C,D), X:obj(C) => F(X):obj(D).
     F:fnctr(C,D) => F:covariant | F:contravariant.
                                                              »
```

The familiar functor rules might be expressed like this

```
                                                              «
     F:fnctr(C,D), X:obj(C), Y:obj(D),
        G:X→Y, F:covariant => F(G):F(X)→F(Y).
     F:fnctr(C,D), X:obj(C), Y:obj(D),
        G:X→Y, F:contravariant => F(G):F(Y)→F(X).
                                                              »
```

Notice that an antecedent literal like 'F:fnctr(C,D)' can occur without explicitly typing C and D. Why?

§4 Autolog natural transformation metalogic

A "natural transformation" is a relation between functors.

```
                                                              «
C:cat, D:Cat,
   F:functor(C,D), G:functor(C,D)
         => nat(F,G):type.   // nat(F,G):type
                                                              »
N:nat(F,G), F:functor(C,D), G:functor(C,D)
        X:obj(C) =>
             N(X):F(X)→G(X).
F:functor(C,D), G:functor(C,D), N:nat(F,G),
        X:obj(C), Y:obj(C),  H:C→C,=>
             N(Y)◦F(H) = G(H)◦N(X).  //   Fig.1
                                                              »
```

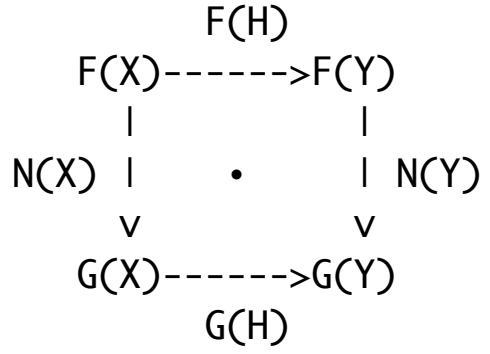

Fig.1

A "natural isomorphism" of covariant functors F and G is a natural transformation N such that for each relevant X, N(X) is an isomorphism.

EXERCISE A. Formulate an autolog definition for natural isomporphism.

.. to be continued

---

§ References

Compare usual definitions for category concepts find here:

1. https://en.wikipedia.org/wiki/Category_theory
2. https://ncatlab.org/nlab/show/category+theory