#5
EXAMPLES OF
SET TYPE METALOGIC
2/15/2020

This lecture poses some possible set-type-logic theory ideas for Autolog, and exercises for the reader to develop.

§1 Set-type-logic autolog terms -explicit comprehension

For a simple example, let's use a small finite type which is explicitly enumerated.

```
«
   true => a:t, b:t, c:t, d:t, e:t, f:t, g:t.   // 1
»
```

Suppose that p:t→prop is a proposition regarding type t.

```
«
   true => p(a), p(c), p(e), p(g).              // 2
»
```

We might specify the set of objects of type t having property p using terms like this ...

```
«
   X:t, p(X) => X∈set(t,p).                    // 3
»
```

Consider the goal

```
«
   g∈set(t,p) => goal.                         // 4
»
```

Here is an autolog proof of the goal ...

```
          true
            |-  rule 1
           g:t
            |-  rule 2
           p(g)
            |-  rule 3
           g∈set(t,p)
            |-  rule 4
           goal
```

EXERCISE 1. Explore how things go IF rule 3 had been
«
```
   X:t, p(X) => X∈set(Z:t,p(Z)).                // 3
```
»
where set(Z:t,p(Z)) is used to look like math notation {Z:T | P(Z)}. The issue is using an autolog variable in the set comprehension term. The set(t,p) notation is more in line with dependent type term notation. By the way, an interesting little project is to add set comprehension terms {Z:T | P(Z)} to autolog.

§2 Generalize set comprehension for types

If we generalize on the type and predicate in rule 3 above we get something like
«
```
   T:type, X:T, P:T→prop, P(X) => X∈set(T,P).   //3a
```
»
Notice that this rule is indexical, predicate P being matched in the predicative literal to the left.

EXERCISE 2. Replace the rule 3 by the the indexical version 3a and explain how a proof obtains.

The converse of 3a is also a useful axiom:

```
«
   T:type, P:T→prop, X∈set(T,P) => X:T, P(X).  //3b
»
```

EXERCISE 3. Construct a proof of the following autolog problem

```
«
   true => int:type, q:int→prop, m∈set(int,q).
   int(m), q(m) => goal.
»
```

§3 Generalize set comprehension for parametric logic

Parametric logic operators can be characterized by rule(s)

```
«
   T:type, P:T→prop, Q:T→prop =>
                            P∧Q:T→prop,
                            PvQ:T→prop,
                             ¬P:T→prop.
»
```

In an appropriate context where T:type, P:T→prop and Q:T→prop, consider the parametric logic equality modulators

```
«
   (P∧Q)(X) = P(X)∧Q(X).
   (PvQ)(X) = P(X)vQ(X).
»
```

The left-hand sides are "parametric" functor expressions because the operators are themselves functors. The term ¬P(X) is read in the same as (¬P)(X), and is also a parametric functor expression.

Similarly, regarding set comprehension, we might employ modulators

```
«
   set(T,P∧Q) = set(T,P) ∩ set(T,Q).
   set(T,P∨Q) = set(T,P) ∪ set(T,Q).
     set(T,⊥) = ∅.
     set(T,⊤) = universe(T). // all X:T
»
```

As in meta lecture #1, we can employ unfolding rules

```
«
   A∧B => A, B.
   A∨B => A | B.
»
```

The sets set(T,P) and set(T,¬P) are considered as abstract complements

The following is an open-ended exercise -- more like a PROJECT.

Exercise 4. The sets

```
        set(T,P)
        set(T,¬P)
```

for T:type and P:T→prop might be considered as "abstract" complement sets. Explore what one might be able to deduce about abstract complement sets using autolog systems, and what the limitations are.