PARADOXES AND FOUNDATIONS

Formally, in set theory a function is just a set, and its application to an element is given by a formula. However, both during translation and for efficient use of FOL theorem proving, we would like to treat functions like functions of predicate calculus. We have a similar ambiguity regarding properties to be treated as sets/classes or as predicates, but the function case is more important as not much is lost in treating properties as sets/classes only.

We explore here the various issues and foundations that protect from these. We fix foundations that protect us from paradox at the cost of a loss of information. However, enough meta-logic should be preserved to translate to Set theory (in First Order Logic) or Higher Order Logic to the extent that it is safe to do so. The meta-logic should also determine, if it is possible, translation to a full set of FOL axioms or to second-order arithmetic.

First we turn to a paradox.

Liberal foundations. Assume that

- We can treat functions as terms, and so apply functions to functions.
- Functions are everywhere defined (perhaps by arbitrarily extending them).
- A function defined on a class of terms can be extended to all terms.
- The degree of a function gives a function to non-negative integers (with value 0 for non-functions).
- Any formula that satisfies the conditions for giving a function gives a function.

Functions have associated to them degrees, so we can regard terms as functions of degree 0, so everything is a function. Introducing a few constants, we can replace predicates and sets by (Boolean) functions.

Constructing functions. A function is given by an explicit or implicit formula. Provided this is *graph-like* on an explicit set given by a formula, we can extend this to a formula giving a function everywhere. Then there is a language extension with a function given by the formula.

A paradox. To obtain a paradox, first consider a function $i(\cdot)$ of degree-one without fixed points. Now define the function g on degree-one functions f by

$$g(f) = i(f(f))$$

and extend to all functions.

This gives a function g. We see that

$$g(g) = i(g(g))$$

contradicting the fact that $i(\cdot)$ has no fixed points.

No paradox in Set theory. We cannot define the function g in set theory, as the graph is defined in a Cantor-like way, and not in terms of powersets etc. A more illuminating viewpoint is to take the defining formula $g(f) = f(f)$, on translating to set theory, to hold for a class of functions f, or better still have the defining property holding within a fixed universe. We then see that $g(g)$ will not be defined.

The obstruction is an implicit typing of sets obtained by their position in the filtering of the Von-Neumann universe, or more naturally a similar inductive construction with a larger base.

No paradox in a Higher-Order Logic. The function g must have a type. Whatever is this type, there is a corresponding type for functions f on which it is defined. For any reasonable type theory, one of the following must happen.

- The expression $f(f)$ is not defined.
- g obtains a type so g is not in its own domain.

Either way the paradox is avoided.

Naive avoiding of paradox. As per the output of the translator, the paradox is avoided in a rather simple way, namely the function $f(\cdot)$ of degree-one and the term f are *different*. This in fact makes the definition $g(f) = i(f(f))$ a meaningless one, as the right-hand side has two free variables, the term f which is also a free variable on the Left-hand side and the function $f()$. In much the same way the quantification ovr functions does not happen as $\forall(f)f(x) = 1$ quantifies over f (of degree 0) but not $f()$ (of degree one).

The cost of avoiding this paradox is that we misinterpret unless we are in First-Order Logic, for example if we define

$$\varphi(f) = f(1) - f(0)$$

this becomes ill-defined due to the same lack of relation between f and $f()$.

Unification foundations. We have a system of relative typing, with inductively constructed data having types with inductively given bounds. Terms may or may not have a given type. The type inference machine should try to define types:

- Consistent with the bounds.
- So that functions are defined, or more generally the assertions to verify are minimal.
- Which unifies as much as possible for the given bounds.

Quotients. It is important to have quotient types, in addition to products and functionals.