Introduction

Mathematics differs from all other sciences by the fact that it provides proofs for its claims. In this course we study what proofs are, and what we can do with them apart from assuring us of the truth of what they state.

Present technology makes it feasible to generate even large formal proofs with "proof assistants". These proof objects can be checked for logical correctness, independently of the used proof assistant: one only needs to check that the logical rules have been applied correctly.

Our underlying "minimal" logic is "constructive" in the sense that a "computationally relevant" (c.r.) predicate (for instance an existential statement $\exists_x A$ or a disjunction $A \vee B$) can only be proved by providing an example. Then from a proof of a c.r. statement we can extract a term which can be seen as a program representing the computational content of the proof. We will develop the theoretical concepts used to formulate and prove this "Soundness theorem".

Usage of a proof assistant makes it possible to automate this extraction process, and also to automatically generate a formal proof that the extracted term "realizes" the original statement. Several applications will be given.

The following list describes some relevant issues. Possibly cryptic notions used above or in these descriptions will be defined in the course.

(1) **Concrete vs. abstract.** To study extraction of programs from proofs it seems best to start with concrete rather than abstract mathematics.

(2) **Types.** Mathematical objects have "types". Examples are "base types" $\mathbb{N}$, $\mathbb{Q}$ of natural or rational numbers, and the "function type" $\mathbb{N} \to \mathbb{N}$ of unary functions f on $\mathbb{N}$. The iteration operator I with $I(f,n,m) := f^{(n)}(m)$ has type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

(3) **Approximations.** Infinite objects are viewed as given by their (finite) approximations. Examples are real numbers, or functions $f \colon \mathbb{N} \to \mathbb{N}$.

(4) **Model.** Objects of type τ are the "partial continuous functionals" of Scott (1982) and Ershov (1977). Continuity of a function f here means that for every approximation V of the value $f(x)$ there is

an approximation U of the argument x such that $f[U]$ has more information than V.

(5) **Constants** of type τ are given by "computation rules". Example: Double: $\mathbb{N} \to \mathbb{N}$ is defined by

$$\mathrm{Double}(0) = 0, \quad \mathrm{Double}(S(n)) = S(S(\mathrm{Double}(n))).$$

(6) **Predicates** of arity $\vec{\tau}$ (a list of types) are either variables or constants. A predicate variable is a place holder for a formula with some object variables bound (for example $\{\, n, m \mid n \leq m \,\}$). A predicate constant is inductively or coinductively defined. This means that it is the least or greatest fixed point of some "clauses". An example of an inductive predicate is Even, the least fixed point of the clauses

$$\mathrm{Even}(0), \quad \forall n(\mathrm{Even}(n) \to \mathrm{Even}(S(S(n)))).$$

An example of a coinductive predicate is ${}^{\mathrm{co}}\mathrm{Even}$, the greatest fixed point of the "closure axiom"

$$\forall n({}^{\mathrm{co}}\mathrm{Even}(n) \to n = 0 \vee \exists_m({}^{\mathrm{co}}\mathrm{Even}(m) \wedge n = S(S(m)))).$$

The logical connectives $\exists$, $\vee$ and $\wedge$ are viewed as inductively defined predicates with parameters.

(7) **Proof trees.** To generate proofs we think of them as trees with "holes" to be filled. Each hole carries a formula (called "goal") and a "context" consisting of object and assumption variables. The task then is to "refine" the proof by filling a hole with a logical rule, which may bind one of the object or assumption variables. This process is well supported by an interactive proof assistant. Guidance by a previous proof idea is of course essential here.

We shall use the proof assistant Minlog `http://www.minlog-system.de` (under development at LMU), which is suitable for the tasks described above. It is implemented in the programming language Scheme (cf. Abelson et al. (1996))