PROOF THEORY AND COMPUTATION

HELMUT SCHWICHTENBERG

Proof theory originated with David Hilbert. He proposed what is now called Hilbert's program, namely to consider formalized proofs as mathematical objects, to be studied in their own right. His main goal was to formally prove consistency of interesting formal theories like arithmetic, in the sense that there is no proof of 0=1. Such consistency proofs cleary have to be carried out in a resticted setting, using only what Hilbert called finitistic methods. Hilbert's program in its original form was refuted by Gödel's well known incompleteness theorems. Gentzen clarified the situation, by showing (i) that it takes transfinite induction up to ε_0 (the least fixed point of ω^α) to prove consistency of arithmetic, and (ii) that transfinite induction up to any $\alpha < \varepsilon_0$ is provable in arithmetic. These results also have a computational aspect: one can define a hierarchy of fast growing functions $(F_\alpha)_{\alpha\leq\varepsilon_0}$ such that each F_α for $\alpha < \varepsilon_0$ is definable in arithmetic, but F_{ε_0} is not (which follows from a majorization argument).

If arithmetic is extended into higher types (functionals etc) one doesn't need ordinals to define the F_α, but can define them in a perspicious way from simple iteration functionals by application alone. Already this observation indicates that from a computational point of view it is of interest to study higher type functionals. We discuss a logical framework TCF (theory of computable functionals) suitable for the extraction of computational content from proofs. TCF can be seen as a variant of HA^ω (Heyting's arithmetic in all finite types). The main differences are (i) TCF has the Scott-Ershov partial continuous functionals as its intended model; (ii) the term part of TCF is an extension T^+ of Gödel's system T with functions defined by possibly non-terminating rules; (iii) (co)inductive predicates with their least (and greatest) fixed point axioms are allowed.

Following Kolmogorov we then view formulas as problems asking for a solution, called "realizers" by Kleene and Kreisel. For this to make sense we first have to define what a realizer for a (co)inductive predicate I applied to some arguments $\vec{r}$ is. The natural choice here is to take a witness of our knowledge that I holds for $\vec{r}$. This is an object in the free algebra build from constructors corresponding to I's defining clauses; for coinductive predicates this object can be infinite (a "stream"). We then can define what the "type" of a formula is, i.e., the type of a solution to the problem posed by the formula. The soundness theorem states that from a proof M of a formula A of type ρ we can extract a term $\mathrm{et}(M)$ in T^+ of the same type ρ such that (provably in TCF) the term $\mathrm{et}(M)$ is a realizer of A.

As a first example we consider a constructive proof of the fan theorem for "coconvex" bars, based on recent work of Josef Berger and Gregor Svindland. Using an appropriate concept of uniform coconvexity and viewing paths as streams, we extract from the formalized proof a simple algorithm computing the upper bound. The second example deals with real number computation based on Gray-code (a binary number system where neighboring values differ in one digit only). Tsuiki has introduced Gray code to the field of real number computation. He assigns to each number a unique $1\bot$-sequence, i.e., an infinite sequence of $\{-1, 1, \bot\}$ containing at most one $\bot$ (meaning undefinedness). Instead of Tsuiki's indeterministic multihead Type-2 machine, we use pre-Gray code, which is a representation of Gray-code as a sequence of constructors, to avoid the difficulty due to $\bot$ which prevents sequential access to a stream. We extract algorithms working with Gray-coded real numbers from proofs formalized in TCF.