Code Carrying Proofs

Helmut Schwichtenberg
Ludwig-Maximilians-Universität, München, Germany

It is well known that constructive (and even some classical) proofs carry algorithmic information. We address the question of how to extract from proofs code executing these algorithms. The advantages are plain: since the programs are derived from machine checkable formalized proofs, there is no question as to their correctness. Moreover, program development (e.g. adaption to changing assumptions on the context or data) can be done on the high level of mathematical proof, rather than on the code level.

We first discuss the general framework we are working in. The logic we use is constructive, or more precisely, the meaning of a formula is given by the Brouwer-Heyting-Kolmogorov understanding of the logical connectives. This involves the notion of a computable functional, which we will define first, via the Scott-Ershov notion of partial continuous functionals.

Next the logical axioms and rules are given, which all have to be valid in the sense above. More technically, we define what it means that a term realizes a formula, and prove a soundness theorem stating that every derivable formula is realizable. In particular this says that any proof of an existential statement contains an algorithm.

As an example for an interesting existential proof we take the well-known argument - involving so-called logical relations - that every simply typed lambda-term has a normal form (i.e., can be evaluated). Since lambda-terms are the kernel of all functional programming languages, this is a rather fundamental result. Its proof will be discussed in detail, and we will see what the general program extraction machinery does in this case.

Recommended Background Reading

1. A. Troelstra; H. Schwichtenberg. *Basic Proof Theory*, Cambridge University Press, 2nd ed. 2000, Ch. 2 and 6

2. U. Berger. *Program extraction from normalization proof*, In M. Bezem and J.F. Groote (eds), Proc. TLCA 1993, Springer LNCS 664, pp.91-106, 1993