Certifying Computations: Algorithmics meets Software Engineering
Kurt Mehlhorn
Max-Planck-Institute for Informatics

I am mostly interested in algorithms for difficult combinatorial and geometric problems: What is the fastest tour from A to B? How to optimally assign jobs to machines? How can a robot move from one location to another one? Algorithms solving such problems are complex and their implementation is error-prone.

Program for $f$

How can we make sure that our implementations of such algorithms are reliable? Certifying algorithms are a viable approach towards the goal. The top part of the figure above illustrates the I/O behavior of a conventional program for computing a function $f$. The user feeds an input $x$ to the program and the program returns an output $y$. Why should the user believe that $y$ is equal to $f(x)$?

A certifying algorithm for $f$ computes $y$ and a witness (proof) $w$; $w$ proves that the algorithm has not erred for this particular input. The certifying algorithms is accompanied by a checker program $C$. It accepts the triple $(x, y, w)$ if and only if $w$ is a valid witness for the equality $y = f(x)$. Certifying algorithms are the design principle for LEDA, the library of efficient data types and algorithms ([MN99]).

In the first part of the talk, we introduce the concept of certifying algorithms and discuss its significance. In the second part of the talk, we survey certifying algorithms ([MMNS11]). In the third part of the talk, we discuss the formal verification of certifying computations ([ABMR14, NRM14]).


